CSC3200 Final Review

Data Structures and Advanced Programming

Chaoyi Sun

CUHK-Shenzhen Programming Contest Team

Dec 8 2025

1 Introduction

2 C++ Basics

3 Data Structures

4 Application

5 Q & A

Self-Introduction

Information

- **Name** Chaoyi Sun
- **Year** Second-year Undergraduate
- **Major** Computer Science

Achievements

- Silver Medal, 50th ICPC Asia Regional Contest (Wuhan)
- Silver Medal, 50th ICPC Asia Regional Contest (Nanjing)
- Bronze Medal, 11th CCPC National Contest (Chongqing)

1. Introduction
2. C++ Basics
   - Variable
   - Memory
   - Pointer
3. Data Structures
4. Application
5. Q & A

1 Introduction

2 C++ Basics
   Variable
   Memory
   Pointer

3 Data Structures

4 Application

5 Q & A

C++ Keywords

alignas	alignof	and	and_eq	asm	auto
bitand	bitor	bool	break	case	catch
char	char8_t	char16_t	char32_t	class	compl
concept	const	consteval	constexpr	constinit	const_cast
continue	co_await	co_return	co_yield	decltype	default
delete	do	double	dynamic_cast	else	enum
explicit	export	extern	false	float	for
friend	goto	if	inline	int	long
mutable	namespace	new	noexcept	not	not_eq
nullptr	operator	or	or_eq	private	protected
public	register	reinterpret_cast	requires	return	short
signed	sizeof	static	static_assert	static_cast	struct
switch	template	this	thread_local	throw	true
try	typedef	typeid	typename	union	unsigned
using	virtual	void	volatile	wchar_t	while
xor	xor_eq				

Operator Precedence

Prec.	Operator	Assoc.
1	() [] -> . :: ++ −− (postfix)	L→R
2	! ∼ ++ −− (prefix) - + * &	R→L
3	->* .*	L→R
4	* / %	L→R
5	+ -	L→R
6	« »	L→R
7	< <= > >=	L→R
8	== !=	L→R
9	&	L→R
10	^	L→R
11	|	L→R
12	&&	L→R
13	||	L→R
14	? :	R→L
15	= += -= *= /= %= &= ^= |= «= »=	R→L
16	,	L→R

1 Introduction

2 C++ Basics
- Variable
- Memory
- Pointer

3 Data Structures

4 Application

5 Q & A

Memory

Memory Region	Characteristics
Stack	Automatic stack frames created for each method call to hold local variables
Heap	Dynamic memory pool for manual allocation and lifetime control.
Dynamic Allocation	**[manual]** `malloc/free`
Global/Static	**[Auto]** Global variables/constants that persist throughout the lifetime of the program.
Constant	**[Auto]** constants/read-only

Example

```
void f1 ()
{
    int *p = new int[5];
    int variable = 10;
}
```

```
0000000000001149 <_Z2f1v>:a
    1149:    f3 0f 1e fa              endbr64
    114d:    55                       push   %rbp
    114e:    48 89 e5                 mov    %rsp,%rbp
    1151:    48 83 ec 10              sub    $0x10,%rsp
    1155:    bf 14 00 00 00           mov    $0x14,%edi
    115a:    e8 f1 fe ff ff           call   1050 <_Znam@plt>
    115f:    48 89 45 f8              mov    %rax,-0x8(%rbp)
    1163:    c7 45 f4 0a 00 00 00     movl   $0xa,-0xc(%rbp)
    116a:    90                       nop
    116b:    c9                       leave
    116c:    c3                       ret
```

Example (Cont.)

```
void f1 ()
{
    int *p = new int[5];
    delete[] p;
}
```

```
0000000000001149 <_Z2f1v>:
    1169:    f3 0f 1e fa             endbr64
    116d:    55                      push   %rbp
    116e:    48 89 e5                mov    %rsp,%rbp
    1171:    48 83 ec 10             sub    $0x10,%rsp
    1175:    bf 14 00 00 00          mov    $0x14,%edi
    117a:    e8 e1 fe ff ff          call   1060 <_Znam@plt>
    117f:    48 89 45 f8             mov    %rax,-0x8(%rbp)
    1183:    48 83 7d f8 00          cmpq   $0x0,-0x8(%rbp)
    1188:    74 0c                   je     1196 <_Z2f1v+0x2d>
    118a:    48 8b 45 f8             mov    -0x8(%rbp),%rax
    118e:    48 89 c7                mov    %rax,%rdi
    1191:    e8 da fe ff ff          call   1070 <_ZdaPv@plt>
    1196:    90                      nop
    1197:    c9                      leave
    1198:    c3                      ret
```

1 Introduction

2 C++ Basics
- Variable
- Memory
- Pointer

3 Data Structures

4 Application

5 Q & A

Pointer / Reference

```cpp
int vari = 3100;
int *ptr = &vari; // pointer
int &ref = vari; // reference
*ptr = 3200;
```

Differences

- **Pointer**: Can be reassigned, can be null, has separate memory.
- **Reference**: Must be initialized, cannot be reassigned, alias to existing variable.

Pointer / Reference

```cpp
int vari = 3100;
int *ptr = &vari; // pointer
int &ref = vari; // reference
*ptr = 3200;
```

Differences

- **Pointer**: Can be reassigned, can be null, has separate memory.
- **Reference**: Must be initialized, cannot be reassigned, alias to existing variable.

The meaning of *

- `int *ptr = &vari;` Type specifier in declarations (store address).
- `*ptr = 3200;` Dereference operator in expressions.

Multi-level Pointer

```
#include <cstdio>
void calc (int a,int b,int **f1,int **f2)
{
    **f1 = a + b;**f2 = a - b;
    int *temp = *f1;*f1 = *f2;*f2 = temp;
}
int main ()
{
    int a = 5,b = 3,x,y;
    int *px = &x,*py = &y;
    calc (a,b,&px,&py);
    printf ("%d %d\n",x,y);
    return 0;
}
```

Multi-level Pointer

```
#include <cstdio>
void calc (int a,int b,int **f1,int **f2)
{
    **f1 = a + b;**f2 = a - b;
    int *temp = *f1;*f1 = *f2;*f2 = temp;
}
int main ()
{
    int a = 5,b = 3,x,y;
    int *px = &x,*py = &y;
    calc (a,b,&px,&py);
    printf ("%d %d\n",x,y);
    return 0;
}
```

Answer

```
8 2
```

Swap

```
#include <cstdio>
void swap1 (int *x,int *y) {int tmp = *x;*x = *y;*y = tmp;}
void swap2 (int *x,int *y) {int *tmp = x;x = y;y = tmp;}
void swap3 (int **x,int **y) {int *tmp = *x;*x = *y;*y = tmp;}
void swap4 (int *&x,int *&y) {int *tmp = x;x = y;y = tmp;}
int main ()
{
    int a = 5,b = 3;
    swap1 (&a,&b);
    printf ("%d %d\n",a,b);
    a = 5,b = 3;
    swap2 (&a,&b);
    printf ("%d %d\n",a,b);
    a = 5,b = 3;
    int *pa = &a,*pb = &b;
    swap3 (&pa,&pb);
    printf ("%d %d %d %d\n",a,b,*pa,*pb);
    a = 5,b = 3;
    pa = &a;pb = &b;
    swap4 (pa,pb);
    printf ("%d %d %d %d\n",a,b,*pa,*pb);
    return 0;
}
```

Swap (cont.)

Answer

```
3 5
5 3
5 3 3 5
5 3 3 5
```

Hint

Drawing memory diagrams can greatly assist in solving such problems!

Array

```
int arr[5] = {0,1,3,4,7};
*(arr + 3) = 5;
for (int i = 0;i < 5;++i) printf ("The %d-th: %d\n",i,*(arr + i));
```

```
The 0-th: 0
The 1-th: 1
The 2-th: 3
The 3-th: 5
The 4-th: 7
```

Array (Cont.)

```
#include <cstdio>
/* declaration here */
int main ()
{
    char arr[2][3] = {{'a','b','c'},{'e','f','g'}};
    calc (arr);
    return 0;
}
```

Which of the following function declarations are correct?

A) `void calc (char (*arr)[3]);`

B) `void calc (char arr[2][3]);`

C) `void calc (char **arr);`

D) `void calc (char arr[][3]);`

Array (Cont.)

```
#include <cstdio>
/* declaration here */
int main ()
{
    char arr[2][3] = {{'a','b','c'},{'e','f','g'}};
    calc (arr);
    return 0;
}
```

Which of the following function declarations are correct?

A) `void calc (char (*arr)[3]);`

B) `void calc (char arr[2][3]);`

C) `void calc (char **arr);`

D) `void calc (char arr[][3]);`

Answer

```
ABD
```

Pointer and Const

- **Const pointer** Pointer itself cannot be changed.

`int* const p1;`

- **Pointer to const** Pointed value cannot be changed.

`int const *p1;` or `const int *p1;`

- **Const pointer to const** Both cannot be changed.

`const int* const p1;`

`const int const *p1;` is wrong due to redundant const qualifier.

Increment/Decrement

```
#include <cstdio>
using namespace std;
int main ()
{
    int a[] = {3,1,4,1,5,9};
    int *p = a + 1,*q = a + 4;
    int x = *++p,y = *q--;
    int z = ++*p,w = (*q)--;
    int t = (*p++) + (*--q);
    int u = *p,v = *q;
    for (int i = 0;i < 6;++i) printf ("%d",a[i]);
    printf ("\n%d%d%d%d%d%d%d\n",x,y,z,w,t,u,v);
    return 0;
}
```

Increment/Decrement

```
#include <cstdio>
using namespace std;
int main ()
{
    int a[] = {3,1,4,1,5,9};
    int *p = a + 1,*q = a + 4;
    int x = *++p,y = *q--;
    int z = ++*p,w = (*q)--;
    int t = (*p++) + (*--q);
    int u = *p,v = *q;
    for (int i = 0;i < 6;++i) printf ("%d",a[i]);
    printf ("\n%d%d%d%d%d%d%d\n",x,y,z,w,t,u,v);
    return 0;
}
```

Answer

```
315059
45511005
```

1. Introduction
2. C++ Basics
3. Data Structures
   - Sequence Containers
   - Associative Containers
   - Tree
4. Application
5. Q & A

1. Introduction
2. C++ Basics
3. Data Structures
   - Sequence Containers
   - Associative Containers
   - Tree
4. Application
5. Q & A

Vector

```cpp
int main ()
{
    vector <int> lst = {2,0,2,5,1,2,1,4};
    int tot = 0,flag = 0,n = lst.size ();
    for (auto it = lst.begin ();it != lst.end ();++it)
    {
        if (flag) tot += lst.front ();
        else tot += *it;
        flag = 1 - flag;
    }
    printf ("%d,%d\n",tot,tot / n);
    return 0;
}
```

Vector

```
int main ()
{
    vector <int> lst = {2,0,2,5,1,2,1,4};
    int tot = 0,flag = 0,n = lst.size ();
    for (auto it = lst.begin ();it != lst.end ();++it)
    {
        if (flag) tot += lst.front ();
        else tot += *it;
        flag = 1 - flag;
    }
    printf ("%d,%d\n",tot,tot / n);
    return 0;
}
```

Answer

```
14,1
```

Vector (Cont.)

resize (),reserve ()

```
vector <int> lst = {2,0,2,5};
lst.reserve (8);
printf ("%ld,%ld,%d\n",lst.size (),lst.capacity (),lst.back ());
lst.reserve (6);
printf ("%ld,%ld,%d\n",lst.size (),lst.capacity (),lst.back ());
lst.resize (10,3);
printf ("%ld,%ld,%d\n",lst.size (),lst.capacity (),lst.back ());
lst.resize (3,5);
printf ("%ld,%ld,%d\n",lst.size (),lst.capacity (),lst.back ());
```

Vector (Cont.)

resize (),reserve ()

```
vector <int> lst = {2,0,2,5};
lst.reserve (8);
printf ("%ld,%ld,%d\n",lst.size (),lst.capacity (),lst.back ());
lst.reserve (6);
printf ("%ld,%ld,%d\n",lst.size (),lst.capacity (),lst.back ());
lst.resize (10,3);
printf ("%ld,%ld,%d\n",lst.size (),lst.capacity (),lst.back ());
lst.resize (3,5);
printf ("%ld,%ld,%d\n",lst.size (),lst.capacity (),lst.back ());
```

Answer

```
4,8,5
4,8,5
10,10,3
3,10,2
```

Stack/Queue

```cpp
#include <stack>
#include <cstdio>
class CSC
{
    private:
    std::stack <int> s1,s2;
    void tran () {while (!s1.empty ()) s2.push (s1.top ()),s1.pop ();}
    public:
    CSC () = default;
    void add (int val) {s1.push (val);}
    int del ()
    {
        if (s2.empty ())
        {
            if (s1.empty ()) return -1;
            tran ();
        }
        int val = s2.top ();s2.pop ();
        return val;
    }
};
```

Stack/Queue (Cont.)

```
int main ()
{
    CSC arr;
    for (int i = 1;i <= 5;++i) arr.add (i);
    for (int i = 0;i <= 5;++i)
    {
        printf ("%d",arr.del ());
        if (i != 5) printf (",");
    }
    printf ("\n");
    return 0;
}
```

Stack/Queue (Cont.)

```
int main ()
{
    CSC arr;
    for (int i = 1;i <= 5;++i) arr.add (i);
    for (int i = 0;i <= 5;++i)
    {
        printf ("%d",arr.del ());
        if (i != 5) printf (",");
    }
    printf ("\n");
    return 0;
}
```

Answer

```
1,2,3,4,5,-1
```

Stack/Queue (Cont.)

Sliding Window

```
void calc ()
{
    int q[1005] = {0},num[] = {1,2,7,3,8,5,2,9};
    int k = 3,n = 8,head = 0,tail = -1;
    for (int i = 0;i < n;++i)
    {
        while (head <= tail && q[head] <= i - k) ++head;
        while (head <= tail && num[q[tail]] <= num[i]) --tail;
        q[++tail] = i;
        if (i >= k - 1) printf ("%d",num[q[head]]);
    }
    printf ("\n");
}
```

Stack/Queue (Cont.)

Sliding Window

```
void calc ()
{
    int q[1005] = {0},num[] = {1,2,7,3,8,5,2,9};
    int k = 3,n = 8,head = 0,tail = -1;
    for (int i = 0;i < n;++i)
    {
        while (head <= tail && q[head] <= i - k) ++head;
        while (head <= tail && num[q[tail]] <= num[i]) --tail;
        q[++tail] = i;
        if (i >= k - 1) printf ("%d",num[q[head]]);
    }
    printf ("\n");
}
```

Answer (Hint: It also can be implemented with `std::deque`.)

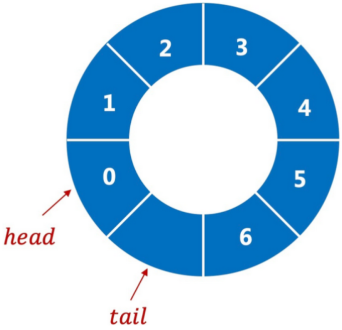
```
778889
```

Stack/Queue (Cont.)

Circular Queue

Empty `head == tail;`

Full `(tail + 1) % capacity == front;`



Link list

Version 1

```
LinkNode* func (LinkNode* head)
{
    LinkNode* pre = nullptr;
    LinkNode* cur = head;
    LinkNode* nxt = nullptr;
    while (cur != nullptr)
    {
        nxt = cur -> next;
        cur -> next = pre;
        pre = cur;cur = nxt;
    }
    return pre;
}
```

Link list (Cont.)

Version 2

```
LinkNode* func (LinkNode* head)
{
    if (head == nullptr || head -> next == nullptr) return head;
    LinkNode* new_head = func (head -> next);
    head -> next -> next = head;
    head -> next = nullptr;
    return new_head;
}
```

1 Introduction

2 C++ Basics

3 Data Structures
- Sequence Containers
- Associative Containers
- Tree

4 Application

5 Q & A

Map

Hashmap

```
#define TABLE_SIZE 7
struct HashTable {int table[TABLE_SIZE];};
int hashFunction (int key) {return key % TABLE_SIZE;}
void insert(int table[],int key)
{
    int id = hashFunction (key);
    while (table[id] != -1) id = (id + 1) % TABLE_SIZE;
    table[id] = key;
}
```

Map (Cont.)

1. The following keys are inserted in order:{7, 14, 21, 28, 35, 42}. After inserting all keys, what is the average successful search length (average number of comparisons to find an existing key)?

   A) 2.0
   B) 2.5
   C) 3.0
   D) 3.5

2. In the worst-case scenario for linear probing, inserting n elements into a table of size n requires?

   A) $O(n)$ time per insertion
   B) $O(\log n)$ time per insertion
   C) $O(n \log n)$ total time for all insertions
   D) $O(n^2)$ total time for all insertions

Map (Cont.)

1. The following keys are inserted in order:{7, 14, 21, 28, 35, 42}. After inserting all keys, what is the average successful search length (average number of comparisons to find an existing key)?

A) `2.0`

B) `2.5`

C) `3.0`

D) `3.5`

2. In the worst-case scenario for linear probing, inserting n elements into a table of size n requires?

A) $O(n)$ time per insertion

B) $O(\log n)$ time per insertion

C) $O(n \log n)$ total time for all insertions

D) $O(n^2)$ total time for all insertions

Answer

```
D;AD
```

Set

Bitset

```
const int RANGE_SIZE = 128;
const int BITS_PER_BYTE = 8;
const int BITS_PER_LONG = BITS_PER_BYTE * sizeof(long);
const int CVEC_WORDS = (RANGE_SIZE + BITS_PER_LONG - 1) / BITS_PER_LONG;
struct BitSet
{
    unsigned long words[CVEC_WORDS];
    BitSet ()
    {
        for (int i = 0;i < CVEC_WORDS;++i) words[i] = 0;
    }
};
unsigned long createMask (int k)
{
    unsigned long x = 1L;
    return x << k % BITS_PER_LONG;
}
```

Set (Cont.)

```
// check if the k-th characters in ASCII table in the Set
bool inSet(BitSet &cv, int k)
{
    if (k < 0 || k >= RANGE_SIZE) return;
    return cv.words[k / BITS_PER_LONG] & createMask (k);
}
// set the k-th character in ASCII table to the set
void setBit(BitSet &cv, int k)
{
    if (k < 0 || k >= RANGE_SIZE) return;
    cv.words[k / BITS_PER_LONG] |= createMask (k);
}
// remove the k-th characters in ASCII Table from the set
void remove(BitSet &cv, int k)
{
    if (k < 0 || k >= RANGE_SIZE) return;
    cv.words[k / BITS_PER_LONG] &= ~createMask (k);
}
```

1 Introduction

2 C++ Basics

3 Data Structures
 - Sequence Containers
 - Associative Containers
 - Tree

4 Application

5 Q & A

Binary Tree

Three Tree Traversal Methods

- **Preorder Traversal** Root → Left → Right
- **Inorder Traversal** Left → Root → Right
- **Postorder Traversal** Left → Right → Root

Question

Preorder traversal 1, 2, 4, 8, 5, 9, 3, 6, 10, 7
Inorder traversal 8, 4, 2, 9, 5, 1, 6, 10, 3, 7
Based on the given preorder and inorder traversals, what is the postorder traversal?

Binary Tree

Three Tree Traversal Methods

- **Preorder Traversal** Root → Left → Right
- **Inorder Traversal** Left → Root → Right
- **Postorder Traversal** Left → Right → Root

Question

Preorder traversal 1, 2, 4, 8, 5, 9, 3, 6, 10, 7
Inorder traversal 8, 4, 2, 9, 5, 1, 6, 10, 3, 7
Based on the given preorder and inorder traversals, what is the postorder traversal?

Answer

8, 4, 9, 5, 2, 10, 6, 7, 3, 1

Binary Tree (Cont.)

```
int find (int x)
{
    for (int i = 0;i < n;++i)
        if (in[i] == x) return i;
    return -1;
}
void getpost (int sl,int sr,int fl,int fr)
{
    if (sl > sr || fl > fr) return;
    int k = find (__1__);
    getpost (__2__);getpost (__3__);
    printf ("%d\n",pre[sl]);
}
void solve () {getpost (0,n - 1,0,n - 1);}
```

Binary Tree (Cont.)

```
int find (int x)
{
    for (int i = 0;i < n;++i)
        if (in[i] == x) return i;
    return -1;
}
void getpost (int sl,int sr,int fl,int fr)
{
    if (sl > sr || fl > fr) return;
    int k = find (__1__);
    getpost (__2__);getpost (__3__);
    printf ("%d\n",pre[sl]);
}
void solve () {getpost (0,n - 1,0,n - 1);}
```

Answer

```
pre[sl]
sl + 1,sl + k - fl,fl,k - 1
sl + k - fl + 1,sr,k + 1,fr
```

BST

```
struct TreeNode
{
    int key,sz,cnt;
    TreeNode *left,*right;
    TreeNode (int val) :
    key (val), sz (1), cnt (1), left (nullptr), right (nullptr) {}
};
```

```
bool search (TreeNode* u,int val)
{
    if (u == nullptr) return false;
    if (u -> key == val) return true;
    else if (val < u -> key) return search (rt -> left,val);
    else return search (rt -> right,val);
}
```

BST (Cont.)

```
int get_sz (TreeNode* u) {return u == nullptr ? 0 : u -> size;}
TreeNode* insert (TreeNode* u, int val)
{
    if (u == nullptr) return new TreeNode (val);
    if (val < u -> key) u -> left = insert (u -> left,val);
    else if (val > u -> key) u -> right = insert (u -> right,val);
    else ++(u -> cnt);
    u -> sz = u -> cnt + get_sz (u -> left) + get_sz (u -> right);
    return u;
}
```

```
TreeNode* findMinNode (TreeNode* u)
{
    while (u -> left != nullptr) u = u -> left;
    return u;
}
```

BST (Cont.)

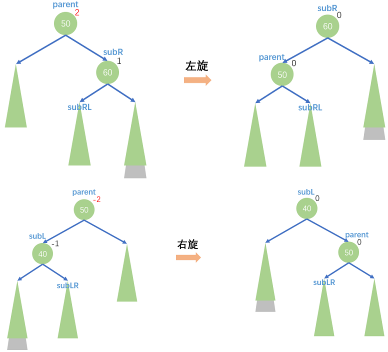
```
TreeNode* remove (TreeNode* u,int val)
{
    if (u == nullptr) return u;
    if (val < u -> key) u -> left = remove (u -> left,val);
    else if (val > u -> key) u -> right = remove (u -> right,val);
    else
    {
        if (u -> cnt > 1) {--(rt -> cnt);return;}
        if (u -> left != nullptr && u -> right != nullptr)
        {
            TreeNode* nxt = findMinNode (u -> right);
            u -> key = nxt -> key;u -> cnt = nxt -> cnt;nxt -> cnt = 1;
            u -> right = remove (u -> right,nxt -> key);
        }
        else
        {
            TreeNode *tmp = u;
            u = (u -> left != nullptr) ? u -> left : u -> right;
            delete tmp;return u;
        }
    }
    u -> sz = u -> cnt + get_sz (u -> left) + get_sz (u -> right);
    return u;
}
```

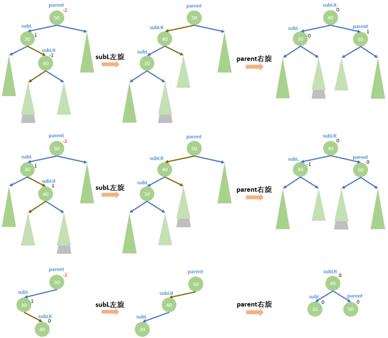
BST (Cont.)

```
int queryRank (TreeNode* u,int val)
{
    if (u == nullptr) return;
    if (v == u -> key) return get_sz (u -> left) + 1;
    else if (v < u -> key) return queryRank (u -> left,val);
    return queryRank (u -> right,val) + get_sz (u -> left) + u -> cnt;
}
```

```
int querykth (TreeNode* u,int k)
{
    if (u == nullptr) return -1;
    if (u -> left)
    {
        if (u -> left -> sz >= k) return querykth (u -> left,k);
        if (u -> left -> sz + u -> cnt >= k) return u -> key;
    }
    else if (k <= u -> cnt) return u -> cnt;
    return querykth (u -> right,k - get_sz (u -> left) - u -> cnt);
}
```

AVL Tree


parent
2
50
subR
1
60
subRL
左旋
subR
0
60
parent
0
50
subRL
parent
-2
50
subL
-1
40
subLR
右旋
subL
0
40
parent
0
50
subLR


parent
-2
50
subL
1
30
subLR
40
subL左旋
parent
50
subLR
40
subL
30
parent右旋
subLR
0
40
subL
30
parent
50
subL左旋
parent右旋
subL左旋
parent右旋

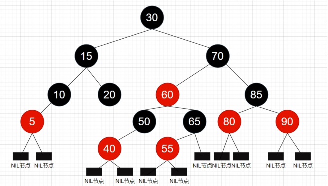
AVL Tree (Cont.)

```
avl_node* R_rotate (avl_node* u)
{
    avl_node* x = u -> L;avl_node* y = x -> R;
    x -> R = u;u -> L = y;
    x -> fat = u -> fat;
    u -> fat = x;
    if (y != nullptr) y -> fat = u;
    u -> upd ();x -> upd ();
    return x;
}
avl_node* L_rotate (avl_node* u)
{
    avl_node* x = u -> R;avl_node* y = x -> L;
    x -> L = u;u -> R = y;
    x -> fat = u -> fat;
    u -> fat = x;
    if (y != nullptr) y -> fat = u;
    u -> upd ();x -> upd ();
    return x;
}
```

AVL Tree (Cont.)

```
avl_node* LR_rotate (avl_node* u) {u -> L = L_rotate (u -> L);return R_rotate (u);}
avl_node* RL_rotate (avl_node* u) {u -> R = R_rotate (u -> R);return L_rotate (u);}
int height (avl_node* u) const {return u ? u -> dep : -1;}
int calc (avl_node* u) const {return !u ? 0 : height (u -> L) - height (u -> R);}
avl_node* balance (avl_node* u)
{
    if (u == nullptr) return nullptr;
    int fac = calc (u);
    if (fac > 1)
    {
        if (calc (u -> L) >= 0) return R_rotate (u);
        u -> L = L_rotate (u -> L);
        return R_rotate (u);
    }
    else if (fac < -1)
    {
        if (calc (u -> R) <= 0) return L_rotate (u);
        u -> R = R_rotate (u -> R);
        return L_rotate (u);
    }
    return u;
}
```

Red-Black Tree



Key Characteristics of RB Trees

- **Binary Search Tree Property**
- **Color Rules**
  - Every node is either red or **black**
  - Root is always **black**
  - All leaves (NIL nodes) are **black**
- **Red Node Constraint**
  - No two consecutive red nodes
- **Black Height Property**
  - Every path from root to leaf has the same number of black nodes
  - Ensures tree remains balanced

Red-Black Tree (Cont.)

Theorem

For a red-black tree with n nodes, its height h satisfies $h \leq 2\log_2(n+1)$.

Proof

Let bh be the black height of a RB tree.
Then we have $n \geq 2^{bh} - 1$ and $h \leq 2bh$.
So we can conclude $h \leq 2bh \leq log_2(n+1)$.

Heap

Heap Construction

$$
\begin{aligned}
T(n) &= 1 \times (H-1) + 2 \times (H-2) + \cdots + 2^{H-1} \times 0 \\
&= 2^H - H - 1 = O(n)
\end{aligned}
$$

```cpp
void down (int *a,int n,int i)
{
    int pos = i,l = 2 * i + 1,r = 2 * i + 2;
    if (l < n && a[l] > a[pos]) pos = l;
    if (r < n && a[r] > a[pos]) pos = i;
    if (pos != i) swap (a[i],a[pos]),down (a,n,pos);
}
void build () //Default: max-heap
{
    for (int i = a.size () / 2 - 1;i >= 0;--i) down (a,a.size (),i);
}
```

Heap (Cont.)

Insert

```
void up (int *a,int id)
{
    int fa = (id - 1) / 2;
    while (id > 0 && a[id] > a[fa])
    {swap (a[id],a[fa]);id = fa;fa = (id - 1) / 2;}
}
void insert (int val)
{
    if (n == _MAX_SIZE) return;
    a[n] = val;++n;up (a,n);
}
```

Delete

```
void del (int *a,int n)
{
    if (n <= 0) return;
    swap (a[0],a[n]);--n;down (a,n,0);
}
```

Heap (Cont.)

Q1. Given the keyword sequence `5, 8, 12, 19, 28, 20, 15, 22` is a **min-heap**, after inserting keyword 3 and performing heap adjustment, the resulting min-heap is:

A) 3, 5, 12, 8, 28, 20, 15, 22, 19

B) 3, 5, 12, 19, 20, 15, 22, 8, 28

C) 3, 8, 12, 5, 20, 15, 22, 28, 19

D) 3, 12, 5, 8, 28, 20, 15, 22, 19

Heap (Cont.)

Q1. Given the keyword sequence 5, 8, 12, 19, 28, 20, 15, 22 is a **min-heap**, after inserting keyword 3 and performing heap adjustment, the resulting min-heap is:

A) 3, 5, 12, 8, 28, 20, 15, 22, 19

B) 3, 5, 12, 19, 20, 15, 22, 8, 28

C) 3, 8, 12, 5, 20, 15, 22, 28, 19

D) 3, 12, 5, 8, 28, 20, 15, 22, 19

Answer

```
A
```

Heap (Cont.)

Q2. Which of the following statements about a **max-heap** (containing at least 2 elements) is/are correct?

I. A heap can be viewed as a **complete binary tree**.

II. A heap can be stored using **sequential (array-based) storage**.

III. A heap can be viewed as a **binary search tree**.

IV. The **second largest element** in the heap must be in the level directly below the root.

A) I and II only

B) II and III only

C) I, II, and IV only

D) I, III, and IV only

Heap (Cont.)

Q2. Which of the following statements about a **max-heap** (containing at least 2 elements) is/are correct?

I. A heap can be viewed as a **complete binary tree**.

II. A heap can be stored using **sequential (array-based) storage**.

III. A heap can be viewed as a **binary search tree**.

IV. The **second largest element** in the heap must be in the level directly below the root.

A) I and II only

B) II and III only

C) I, II, and IV only

D) I, III, and IV only

Answer

```
C
```

1. Introduction
2. C++ Basics
3. Data Structures
4. Application
   - Complexity
   - Sort
   - Graph Traversal
   - Shortest-Path
   - Topological Sort
5. Q & A

1. Introduction
2. C++ Basics
3. Data Structures
4. Application
   - Complexity
   - Sort
   - Graph Traversal
   - Shortest-Path
   - Topological Sort
5. Q & A

Complexity

Complexity	Name	Example
$O(1)$	Constant	Array access
$O(\log n)$	Logarithmic	Binary search
$O(n)$	Linear	Linear search
$O(n \log n)$	Log-linear	Merge sort
$O(n^2)$	Quadratic	Bubble sort
$O(n^3)$	Cubic	Matrix multiplication
$O(2^n)$	Exponential	Subset enumeration

Growth Rates:

$$O(1) \leq O(\log n) \leq O(n) \leq O(n \log n) \leq O(n^2) \leq O(n^3) \leq O(2^n)$$

Complexity (Cont.)

Time Complexity Analysis

```
for (int i = 1;i <= n;++i)
    for (int j = 1;j <= n / 2;++j) ++cnt;
for (int i = 1;i <= n;i *= 2)
    for (int j = 1;j <= n;++j) ++cnt;
for (int i = 1;i <= n;i *= 2)
    for (int j = 1;j <= i;++j) ++cnt;
for (int i = 1;i <= n;++i)
    for (int j = 1;j <= n / i;++j) ++cnt;
```

Complexity (Cont.)

Time Complexity Analysis

```
for (int i = 1;i <= n;++i)
    for (int j = 1;j <= n / 2;++j) ++cnt;
for (int i = 1;i <= n;i *= 2)
    for (int j = 1;j <= n;++j) ++cnt;
for (int i = 1;i <= n;i *= 2)
    for (int j = 1;j <= i;++j) ++cnt;
for (int i = 1;i <= n;++i)
    for (int j = 1;j <= n / i;++j) ++cnt;
```

Answer

$O(n^2)$, $O(n \log n)$, $O(n)$, $O(n \log n)$

1. Introduction
2. C++ Basics
3. Data Structures
4. Application
   - Complexity
   - Sort
   - Graph Traversal
   - Shortest-Path
   - Topological Sort
5. Q & A

Sort

```
void Bubble_Sort () {
    for (int i = 0; i < n;++i)
        for (int j = 0;j < n - i - 1;++j)
            if (a[j] > a[j + 1]) swap (a[j],a[j + 1]);
}
void Selection_Sort () {
    for (int i = 0;i < n;++i)
    {
        int minIndex = i;
        for (int j = i + 1;j < n;++j)
            if (a[j] < a[minIndex]) minIndex = j;
        swap (a[minIndex],a[i]);
    }
}
void Insertion_Sort () {
    for (int i = 1;i < n;++i)
    {
        int key = a[i],j = i - 1;
        while (j >= 0 && a[j] > key) a[j + 1] = a[j],--j;
        a[j + 1] = key;
    }
}
```

Sort (Cont.)

```
template <typename T>
void Merge_Sort (T a[],int l,int r)
{
    if (l >= r) return;
    int mid = (l + r) / 2;
    Merge_Sort (a,l,mid);Merge_Sort (a,mid + 1,r);
    vector <T> lst;
    int posl = l,posr = mid + 1;
    while (posl <= mid && posr <= r)
    {
        if (a[posl] <= a[posr]) lst.push_back (a[posl++]);
        else lst.push_back (a[posr++]);
    }
    while (posl <= mid) lst.push_back (a[posl++]);
    while (posr <= r) lst.push_back (a[posr++]);
    for (int i = l;i <= r;++i) a[i] = lst[i - l];
}
Merge_Sort (a,0,n - 1);
```

Sort (Cont.)

```
template <typename T>
int partition (T a[],int l,int r)
{
    int pivot = a[l];
    while (l < r)
    {
        while (l < r && pivot <= a[r]) --r;
        a[l] = a[r];
        while (l < r && a[l] <= pivot) ++l;
        a[r] = a[l];
    }
    a[l] = pivot;
    return l;
}
template <typename T>
void Quick_Sort (T a[],int l,int r)
{
    if (l >= r) return;
    int pivot = partition (a,l,r);
    Quick_Sort (a,l,pivot - 1);Quick_Sort (a,pivot + 1,r);
}
Quick_Sort (a,0,n - 1);
```

Sort (Cont.)

The Complexity of Different Sort Algorithms

Algorithm	Best	Average	Worst
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
BST Tree Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$

1 Introduction

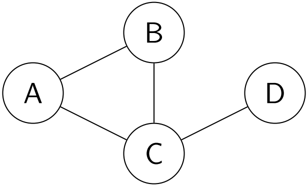
2 C++ Basics

3 Data Structures

4 Application
- Complexity
- Sort
- Graph Traversal
- Shortest-Path
- Topological Sort

5 Q & A

Graph



Adjacency List : $O(|V| + |E|)$.

A: $\rightarrow$ B $\rightarrow$ C $\rightarrow$ $\times$

B: $\rightarrow$ A $\rightarrow$ C $\rightarrow$ $\times$

C: $\rightarrow$ A $\rightarrow$ B $\rightarrow$ D $\rightarrow$ $\times$

D: $\rightarrow$ C $\rightarrow$ $\times$

Adjacency Matrix : $O(|V|^2)$.

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

DFS/BFS

Visualization

Algorithm 1 DFS

procedure SEARCH(u)
  mark u as vis
  for each neighbor v of u **do**
    if v is not vis **then**
      SERACH(v)
    end if
  end for
end procedure

Algorithm 2 BFS

procedure SEARCH($source$)
  $Q \leftarrow$ new Queue()
  Q.enqueue($source$)
  while Q not empty **do**
    $u \leftarrow Q$.dequeue()
    for each neighbor v of u **do**
      if v is not vis **then**
        mark v as vis
        Q.enqueue(v)
      end if
    end for
  end while
end procedure

1 Introduction

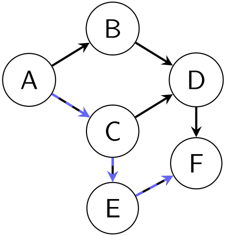
2 C++ Basics

3 Data Structures

4 Application
- Complexity
- Sort
- Graph Traversal
- Shortest-Path
- Topological Sort

5 Q & A

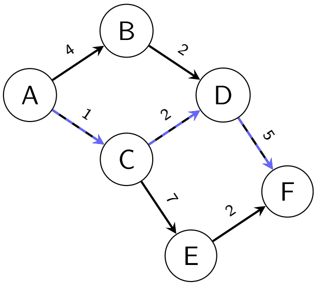
Unweighted Shortest-Path



Algorithm 3 0-1 BFS

```
procedure CALC(source, graph)
    Q ← new Queue()
    dist ← new Dictionary()
    Q.enqueue(source)
    dist[source] ← 0
    while Q is not empty do
        u ← Q.dequeue()
        for each neighbor v of u in graph do
            if v is not in dist then
                dist[v] ← dist[u] + 1
                Q.enqueue(v)
            end if
        end for
    end while
    return dist
end procedure
```

Weighted Shortest-Path



Algorithm 4 Dijkstra

procedure CALC(*source*, *graph*)
    $dist \leftarrow$ array of size $|V|$, initialized to ∞
    $vis \leftarrow$ boolean array of size $|V|$, initialized to false
    $dist[source] \leftarrow 0$
    for $i \leftarrow 1$ to $|V|$ **do**
        $u \leftarrow$ vertex with minimum *dist* not vis
        $vis[u] \leftarrow$ true
        for each neighbor v of u with weight w **do**
            if not $vis[v]$ and $dist[u] + w < dist[v]$ **then**
                $dist[v] \leftarrow dist[u] + w$
            end if
        end for
    end for
    return *dist*
end procedure

1 Introduction

2 C++ Basics

3 Data Structures

4 Application
- Complexity
- Sort
- Graph Traversal
- Shortest-Path
- Topological Sort

5 Q & A

Topological Sort

Algorithm 5 Topological Sort

procedure $\textsc{Search}(G(V, E))$
    $indeg[|V|] \leftarrow 0,\ q \leftarrow \emptyset,\ ord \leftarrow \emptyset$
    for $(u, v) \in E$ **do** $indeg[v] \leftarrow indeg[v] + 1$
    end for
    for $v \in V$ **do**
        if $indeg[v] = 0$ **then** $q.\text{push}(v)$
        end if
    end for
    while $q \neq \emptyset$ **do**
        $u \leftarrow q.\text{pop}(),\ ord.\text{push}(u)$
        for $v \in adj[u]$ **do**
            $indeg[v] \leftarrow indeg[v] - 1$
            if $indeg[v] = 0$ **then** $q.\text{push}(v)$
            end if
        end for
    end while
    if $|ord| \neq |V|$ **then return** $\emptyset$
    end if
    return ord
end procedure

1 Introduction

2 C++ Basics

3 Data Structures

4 Application

5 Q & A

Q & A

Q & A

Thanks for listening!

Good Luck on Your Final Exams!