ÔN TẬP TRẮC NGHIỆM

Contents

1. PHẦN ĐỘ PHỨC TẠP THUẬT TOÁN

1. Độ phức tạp thời gian của thuật toán tìm kiếm tuyến tính là:
   A. O(1)
   B. O(log n)
   C. O(n)
   D. O(n log n)

2. Trong Big-O, ký hiệu **$O(n^2)$** có nghĩa là:
   A. Không gian cần tăng theo n
   B. Tối đa có 2 phép toán
   C. Thời gian tỷ lệ với bình phương của đầu vào
   D. Chỉ áp dụng cho thuật toán đệ quy

3. Cho đoạn mã:

```
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        cout << i << j;
```

Độ phức tạp là:
   A. O(n)
   B. O(log n)
   C. $O(n^2)$
   D. O(n log n)

4. Độ phức tạp của thuật toán đệ quy:

```
void f(int n) {
  if (n <= 1) return;
  f(n - 1);
  f(n - 1);
```

}

là:

A. O(n)

B. O(n log n)

C. O(2^n)

D. O(n^2)

5. Hàm mergeSort có độ phức tạp:

A. O(n)

B. O(n log n)

C. O(n^2)

D. O(log n)

6. Trong trường hợp tốt nhất, insertion sort chạy với độ phức tạp:

A. O(n)

B. O(n^2)

C. O(log n)

D. O(1)

7. Đệ quy có điều kiện dừng sai sẽ gây ra:

A. Vòng lặp vô hạn

B. Tràn ngăn xếp (stack overflow)

C. Lỗi biên dịch

D. Chương trình chạy nhanh hơn

8. Thuật toán nào sau đây có **độ phức tạp trung bình thấp nhất**?

A. Bubble Sort

B. Insertion Sort

C. Merge Sort

D. Selection Sort

9. Độ phức tạp không gian của thuật toán merge sort là:

A. O(1)

B. O(n)

C. O(log n)

D. O(n^2)

10. Với n phần tử, số phép so sánh tối đa trong binary search là:

A. n

B. $\log_2(n)$

C. n log n

D. $\sqrt{n}$

11. Xét đoạn mã sau:

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        // thực hiện thao tác hằng số
    }
}
```

Thời gian chạy của đoạn mã này là:

A. O(n)

B. O(n log n)

C. O(n^2)

D. O(2^n)

12. Xét đoạn mã đệ quy sau:

```
void rec(int n) {
    if (n <= 1) return;
    rec(n - 1);
    rec(n - 1);
}
```

Độ phức tạp thời gian của hàm rec là:

A. O(n)

B. O(n log n)

C. O(2^n)

D. O(n!)

13. Xét đoạn mã sau:

```
for (int i = 1; i < n; i *= 2) {
    // thao tác hằng số
}
```

Độ phức tạp thời gian của vòng lặp này là:

A. O(n)

B. O(log n)

C. O(n log n)

D. O(1)

14. Xét đoạn mã sau:

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        if (i == j) break;
        // thao tác hằng số
    }
}
```

Độ phức tạp thời gian của đoạn mã này là:

A. O(n)

B. O(n log n)

C. O(n^2)

D. O(n^3)

15. Xét đoạn mã đệ quy sau:

```
void func(int n) {
    if(n <= 1) return;
    func(n/2);
    func(n/2);
}
```

Công thức truy hồi của hàm là T(n) = 2T(n/2) + O(1). Độ phức tạp thời gian của hàm là:

A. O(n)

B. O(log n)

C. O(n log n)

D. O(n^2)

2. PHẦN SẮP XẾP & TÌM KIẾM

1. Thuật toán sắp xếp nào **ổn định**?

   A. Selection Sort

   B. Heap Sort

   C. Insertion Sort

   D. Quick Sort

2. Tìm kiếm nhị phân yêu cầu:

   A. Mảng chứa số nguyên

   B. Mảng đã được sắp xếp

   C. Mảng có phần tử trùng nhau

   D. Mảng ngẫu nhiên

3. Thuật toán **Quick Sort** hoạt động tốt nhất khi:

   A. Mảng đảo ngược

   B. Mảng đã sắp xếp

   C. Chia đều 2 nửa

   D. Mảng toàn phần tử giống nhau

4. Trong thuật toán tìm kiếm tuyến tính, thời gian tìm kiếm trong trường hợp xấu nhất là:

   A. O(1)

   B. O(n)

   C. O(log n)

D. O(n log n)

5. Hàm partition() thuộc thuật toán:

   A. Merge Sort

   B. Quick Sort

   C. Heap Sort

   D. Insertion Sort

6. Thuật toán sắp xếp **không đệ quy** là:

   A. Merge Sort

   B. Selection Sort

   C. Quick Sort

   D. Timsort

7. Trường hợp **tốt nhất** của Binary Search là:

   A. O(n)

   B. O(1)

   C. O(n/2)

   D. O(log n)

8. Thuật toán sắp xếp nào có thể đạt hiệu năng O(n) trong trường hợp đặc biệt?

   A. Counting Sort

   B. Merge Sort

   C. Quick Sort

   D. Insertion Sort

9. Ưu điểm lớn nhất của Merge Sort so với Quick Sort là:

   A. Ít phép so sánh hơn

   B. Ít sử dụng bộ nhớ

   C. Đảm bảo O(n log n) mọi trường hợp

   D. Thực thi nhanh hơn

10. Trong Selection Sort, sau mỗi vòng lặp:

A. Mảng được sắp xếp hoàn toàn

B. Một phần tử đúng vị trí

C. Không có gì thay đổi

D. Cần chèn thêm phần tử

11. Xét đoạn mã sau (phân hoạch trong Quick Sort):

```
int a[] = {9, 3, 7, 6, 2, 8};
int pivot = a[0];
int i = 1, j = 5;
while(i <= j) {
    while(i <= j && a[i] <= pivot) i++;
    while(i <= j && a[j] > pivot) j--;
    if(i < j) { int temp = a[i]; a[i] = a[j]; a[j] = temp; }
}
int temp = a[0]; a[0] = a[j]; a[j] = temp;
```

Sau khi thực hiện phân hoạch, vị trí của pivot (giá trị 9) là:

A. 2

B. 3

C. 4

D. 5

12. Xét đoạn mã tìm kiếm nhị phân dưới đây:

```
int a[] = {2, 4, 6, 8, 10};
int key = 6;
int low = 0, high = 4, mid;
while(low <= high) {
    mid = (low + high) / 2;
    if(a[mid] == key) break;
    else if(a[mid] < key) low = mid + 1;
    else high = mid - 1;
}
printf("%d", mid);
```

Giá trị được in ra là:

A. 0

B. 1

C. 2

D. 3

13. Xét đoạn mã sau:

```
void sort(int a[], int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n - 1; j++) {
```

```
            if (a[j] > a[j + 1]) {
                swap(a[j], a[j + 1]);
            }
        }
    }
}
```

Mã trên dùng để sắp xếp tăng dần nhưng không đúng. Lỗi ở đâu?

A. Vòng lặp ngoài cần chạy đến n-1 thay vì n

B. So sánh sai điều kiện a[j] < a[j+1]

C. Giới hạn vòng lặp trong chưa giảm dần (j < n - i - 1)

D. Không có lỗi nào, mã đúng

14. Xét đoạn mã sau dùng để chèn một phần tử x vào mảng tăng dần a có n phần tử:

```
int i = n - 1;
while (i >= 0 && a[i] > x) {
    a[i + 1] = a[i];
    i--;
}
a[i + 1] = x;
n++;
```

Sau đoạn mã, mảng a có đặc điểm gì?

A. Vẫn được sắp xếp tăng dần

B. Mảng bị sai thứ tự

C. Phần tử x luôn đứng đầu

D. Phần tử x luôn đứng cuối

15. Xét mã sau dùng để thực hiện Quick Sort:

```
void quickSort(int a[], int left, int right) {
    int i = left, j = right;
    int pivot = a[(left + right)/2];
    while(i <= j) {
        while(a[i] < pivot) i++;
        while(a[j] > pivot) j--;
        if(i <= j) {
            swap(a[i], a[j]);
            i++; j--;
        }
    }
    quickSort(a, left, j);
    quickSort(a, i, right);
}
```

Mã trên sẽ gây lỗi vì:

A. Không cập nhật pivot khi gọi đệ quy

B. Không có swap() đúng cách

C. Thiếu điều kiện dừng đệ quy (left < right)

D. Không chia mảng đúng cách

16. Cho mảng tăng dần a[] = {1, 2, 4, 4, 4, 5, 6}, đoạn mã sau được dùng:

```
int binarySearch(int a[], int n, int x) {
    int low = 0, high = n - 1, res = -1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (a[mid] == x) {
            res = mid;
            high = mid - 1;
        }
        else if (a[mid] > x) high = mid - 1;
        else low = mid + 1;
    }
    return res;
}
```

Mã trên trả về:

A. Vị trí bất kỳ của x

B. Vị trí cuối cùng của x

C. Vị trí xuất hiện đầu tiên của x

D. Luôn trả về -1

17. Xét đoạn mã:

```
void sortEvenFirst(int a[], int n) {
    int i = 0;
    for (int j = 0; j < n; j++) {
        if (a[j] % 2 == 0) {
            swap(a[i], a[j]);
            i++;
        }
    }
}
```

Sau khi thực hiện, mảng a có tính chất gì?

A. Các số chẵn được sắp xếp đầu mảng, theo thứ tự tăng dần

B. Các số chẵn được đưa lên đầu, không đảm bảo thứ tự

C. Mảng được sắp tăng dần hoàn toàn

D. Các số lẻ bị xóa khỏi mảng

3. DANH SÁCH LIÊN KẾT

1. DSLK đơn có thể duyệt:

A. Từ đầu đến cuối

B. Cả hai chiều

C. Ngẫu nhiên

D. Từ cuối đến đầu

2. DSLK đôi có bao nhiêu con trỏ trong mỗi nút?
   A. 1
   B. 2
   C. 3
   D. Không có

3. Để thêm nút vào đầu DSLK, ta cần:
   A. Duyệt tới cuối
   B. Tạo nút và cập nhật head
   C. Duyệt giữa danh sách
   D. Không cần thao tác gì

4. Độ phức tạp thêm 1 phần tử vào cuối DSLK đơn là:
   A. O(1)
   B. O(n) nếu không có tail
   C. O(log n)
   D. O(n log n)

5. DSLK tròn (circular linked list) là danh sách có:
   A. Phần tử đầu và cuối giống nhau
   B. Nút cuối trỏ về đầu
   C. Hai chiều liên kết
   D. Không có head

6. Trong danh sách liên kết, nếu ta xóa một nút giữa danh sách, độ phức tạp là:
   A. O(n)
   B. O(log n)
   C. O(1)
   D. $O(n^2)$

7. DSLK đôi dễ dàng hơn trong việc:
   A. Chèn vào đầu
   B. Duyệt ngược
   C. Xóa đầu danh sách
   D. Tìm phần tử cuối

8. Sử dụng DSLK sẽ:
   A. Tốn bộ nhớ cho con trỏ
   B. Nhanh hơn mảng trong mọi trường hợp
   C. Không cần cấp phát động

D. Không hỗ trợ duyệt

9. DSLK được ưa chuộng khi:

A. Dữ liệu cố định

B. Dữ liệu thêm/xóa liên tục

C. Cần truy cập ngẫu nhiên

D. Cần sắp xếp nhanh

10. Để duyệt danh sách liên kết cần:

A. Biến con trỏ

B. Biến nguyên

C. Biến chuỗi

D. Không cần gì cả

11. Xét đoạn mã sau:

```
void append(Node* head, int x) {
    Node* newNode = new Node{x, NULL};
    if (head == NULL) {
        head = newNode;
        return;
    }
    Node* p = head;
    while (p->next != NULL)
        p = p->next;
    p->next = newNode;
}
```

Hỏi: **Lỗi sai trong đoạn mã là gì?**

A. Không cấp phát động cho newNode

B. Thêm phần tử vào danh sách rỗng không có hiệu lực vì head là tham trị

C. Không kiểm tra tràn bộ nhớ

D. Duyệt sai danh sách

12. Xét đoạn mã sau:

```
void insertAfter(Node* p, int x) {
    Node* newNode = new Node{x, NULL, p->next};
    p->next = newNode;
    if (newNode->next != NULL)
        newNode->next->prev = newNode;
}
```

Hỏi: **Lỗi sai là gì?**

A. Không cập nhật prev của newNode

B. Không gán newNode->prev = p; nên liên kết ngược bị mất

C. Thiếu điều kiện kiểm tra NULL cho p

D. Gán nhầm next cho p thay vì newNode

13. Xét đoạn mã:

```
void deleteAfter(Node* p) {
    Node* temp = p->next;
    p->next = temp->next;
    delete temp;
}
```

Hỏi: **Trong trường hợp nào đoạn mã trên có thể gây lỗi?**

A. Khi temp không phải nút cuối

B. Khi p == NULL hoặc p->next == NULL

C. Khi danh sách có nhiều hơn 2 phần tử

D. Khi p là nút đầu tiên

14. Giả sử con trỏ p đang trỏ đến đầu DSLK, vòng lặp đúng để duyệt qua toàn bộ danh sách là:

```
while (__________) {
    // xử lý p->data
    p = p->next;
}
```

Chọn đáp án đúng:

A. p == NULL

B. p != NULL

C. p->next != NULL

D. p->data != NULL

15. Giả sử bạn đã tìm được con trỏ prev trỏ đến **nút đứng trước** nút cần xóa trong DSLK. Câu lệnh đúng để **xóa nút kế tiếp** là:

```
Node* temp = prev->next;
prev->next = ________;
delete temp;
```

Chọn đáp án đúng:

A. temp->next->next

B. NULL

C. temp->next

D. prev

4. STACK & QUEUE

1. Stack hoạt động theo nguyên lý:

   A. LIFO

   B. FIFO

   C. FILO

   D. Random

2. Queue hoạt động theo nguyên lý:

   A. LIFO

   B. FIFO

C. LIFO + FIFO
D. Không xác định

3. Hàm push() dùng cho:
A. Stack
B. Queue
C. Deque
D. Linked List

4. Trong Queue, thao tác thêm phần tử gọi là:
A. Pop
B. Enqueue
C. Push
D. Insert

5. Stack có thể dùng để:
A. Duyệt cây
B. Đánh giá biểu thức hậu tố
C. Sắp xếp nhanh
D. Tìm kiếm nhị phân

6. Queue dùng tốt cho:
A. Biểu thức toán học
B. Mô phỏng hàng đợi
C. Kiểm tra chuỗi ngoặc
D. Lưu trữ đệ quy

7. Nếu gọi pop() khi stack rỗng sẽ:
A. Trả về -1
B. Gây lỗi (stack underflow)
C. Không sao
D. Trả về NULL

8. Queue có thể cài đặt bằng:
A. Mảng
B. Danh sách liên kết
C. Cả A và B
D. Không thể cài đặt

9. Deque là:
A. Stack mở rộng

B. Hàng đợi hai đầu

C. Biến thể danh sách liên kết

D. Bộ nhớ động

10. Biểu thức trung tố chuyển sang hậu tố dùng:

A. Stack

B. Queue

C. Deque

D. Cây

11. Cho đoạn code:

```
int stack[100];
int top = -1;

void push(int x) {
    top++;
    stack[top] = x;
}

int pop() {
    if (top == -1) return -1;
    return stack[top--];
}
```

Hỏi: Đoạn mã trên **thiếu gì để tránh lỗi tràn ngăn xếp (stack overflow)?**

A. Kiểm tra top == -1 trong push()

B. Kiểm tra top == 0 trong pop()

C. Kiểm tra top < MAX - 1 trước khi push()

D. Sử dụng stack[top++] = x thay vì top++

12. Cho đoạn code:

```
void printQueue(queue<int> q) {
    while (!q.empty()) {
        cout << q.front() << " ";
    }
}
```

Lỗi của hàm trên là gì?

A. Không in được phần tử cuối

B. Không có lỗi, in đúng

C. Sử dụng q.back() thay vì q.front()

D. Quên gọi q.pop(), gây vòng lặp vô hạn

13. Cho đoạn code:

```
#define MAX 100
int front = 0, rear = 0;
int queue[MAX];

void enqueue(int x) {
    if (rear == MAX) return;
    queue[rear++] = x;
```

```
}

int dequeue() {
    if (front == rear) return -1;
    return queue[front++];
}
```

Hỏi: Lỗi của đoạn mã trên sẽ xảy ra khi nào?

A. Khi hàng đợi vượt quá 100 phần tử

B. Khi enqueue-dequeue liên tục làm rear == MAX dù queue vẫn rỗng

C. Khi front == rear == 0

D. Không có lỗi nếu chỉ dùng ít phần tử

14. Giả sử bạn cần viết hàm đảo ngược chuỗi dùng Stack (giả định stack<char> s; là ngăn xếp ký tự).

```
for (int i = 0; i < str.length(); i++)
    s.push(str[i]);

for (int i = 0; i < str.length(); i++) {
    str[i] = ____________;
}
```

Điền vào chỗ trống:

A. s.front()

B. s.pop();

C. s.top(); s.pop();

D. str[i]

15. Cho đoạn code tính giá trị biểu thức hậu tố:

```
#define MAX 100

int stack[MAX];
int top = -1;

int pop() {
    if (top == -1) return -1;
    return stack[top--];
}

void push(int x) {
    if (top == MAX - 1) return;
    stack[++top] = x;
}

int evaluatePostfix(char expr[]) {
    for (int i = 0; expr[i] != '\0'; i++) {
        char c = expr[i];
        if (c >= '0' && c <= '9') {
            push(c - '0');
        } else if (c == '+' || c == '-' || c == '*' || c == '/') {
            int b = pop();
            int a = pop();
            int result;
            if (c == '+') result = a + b;
            else if (c == '-') result = a - b;
            else if (c == '*') result = a * b;
            else result = a / b;
            push(__________);
        }
```

```
    }
    return pop();
}
```

Điền vào chỗ trống:

A. a + b

B. b

C. result

D. stack[top]

5. CÂY TÌM KIẾM

1. Cây nhị phân tìm kiếm (BST) là cây có:
   A. Nút trái < nút gốc < nút phải
   B. Nút trái = nút phải
   C. Nút gốc nhỏ nhất
   D. Tất cả các nút đều nhỏ hơn gốc
2. Duyệt in-order trong BST trả về:
   A. Ngẫu nhiên
   B. Danh sách tăng dần
   C. Danh sách giảm dần
   D. Dạng cây
3. Chèn một phần tử vào BST mất thời gian:
   A. O(h), với h là chiều cao cây
   B. O(1)
   C. O(n) luôn
   D. O(log n)
4. Cây cân bằng nhất có chiều cao:
   A. n
   B. $\log_2(n)$
   C. √n
   D. 2n
5. Trong BST, tìm phần tử nhỏ nhất:
   A. Đi hết cây phải
   B. Đi về trái đến khi NULL
   C. So sánh mọi nút
   D. Không thể tìm được
6. Xóa một nút trong BST có hai con:
   A. Xóa luôn
   B. Thay bằng giá trị nhỏ nhất bên phải
   C. Không thể xóa
   D. Thay bằng giá trị bất kỳ
7. Cây tìm kiếm nhị phân **có thể** trở thành:

A. Cây đầy

B. Danh sách liên kết nếu thêm sai thứ tự

C. Cây cân bằng

D. Cây AVL

8. Độ phức tạp tìm kiếm trung bình trong BST:

A. O(n)

B. O(log n)

C. $O(n^2)$

D. O(1)

9. Một cây nhị phân tìm kiếm có n nút có bao nhiêu lá tối đa?

A. n

B. n/2

C. log n

D. 1

10. Để duyệt cây theo level (từng tầng), dùng cấu trúc:

A. Stack

B. Queue

C. Deque

D. Mảng

11. Cho đoạn code:

```
int height(Node* root) {
    if (root == NULL) return 0;
    int left = height(root->left);
    int right = height(root->right);
    return max(left, right);
}
```

Hỏi: Kết quả của hàm height() trả về gì?

A. Số lượng nút

B. Chiều cao cây đúng

C. Sai 1 đơn vị so với chiều cao thật

D. Luôn trả về 0 nếu có nút lá

12. Cho root là cây nhị phân tìm kiếm:

```
void inOrder(Node* root) {
    if (root == NULL) return;
    inOrder(root->right);
    cout << root->data << " ";
    inOrder(root->left);
}
```

Duyệt cây theo cách này cho kết quả gì?

A. Duyệt theo thứ tự tăng dần

B. Duyệt theo thứ tự giảm dần

C. Duyệt theo level

D. Không duyệt được cây rỗng

13. Cho đoạn code:

```
bool isBST(Node* root) {
    if (root == NULL) return true;
    if (root->left && root->left->data >= root->data) return false;
    if (root->right && root->right->data <= root->data) return false;
    return isBST(root->left) && isBST(root->right);
}
```

Hàm isBST() trên có lỗi gì?

A. Sai cú pháp

B. Chỉ kiểm tra trực tiếp, không xét giá trị toàn nhánh trái/phải

C. Không duyệt cây con phải

D. Lỗi tràn bộ nhớ

14. Cho đoạn code Tìm phần tử nhỏ nhất trong BST:

```
int findMin(Node* root) {
    while (root->left != NULL) {
        root = ___________;
    }
    return root->data;
}
```

Điền vào chỗ trống:

A. root->right

B. root->left

C. root->data

D. root++

15. Cho đoạn code duyệt cây theo hậu tố

```
void postOrder(Node* root) {
    if (root == NULL) return;
    postOrder(root->left);
    postOrder(root->right);
    ___________;
}
```

Điền vào chỗ trống:

A. postOrder(root->data);

B. cout << root->data << " ";

C. return root->data;

D. root = root->left;

--- Hết ---