

CS 14 - Summer 2004 - Midterm

August 26, 2004

1 Multiple Choice

Fill in the bubble of the single-best answer for each question.

Do NOT make any unrequested marks on your answer sheet. You may use pen or pencil, but if you want to change an answer, you MUST erase completely. Marks that are "X"ed out will be counted as filled.

1. A *stack* is a FIFO.
 - (a) True
 - (b) False
2. Mathematically, if f is $O(n^2)$, then f is also $O(n^3)$.
 - (a) True
 - (b) False
3. Mathematically, if f is $O(n^3)$, then f is also $O(n^2)$.
 - (a) True
 - (b) False
4. Merge sort runs faster if the input is already sorted
 - (a) True
 - (b) False
5. Binary search has the same runtime on a sorted array as on a sorted linked-list.
 - (a) True
 - (b) False
6. A *stack* built using ADT List adds to a different end of the list than it removes from.
 - (a) True
 - (b) False

7. Quick sort has a worst case runtime of $O(n \log n)$
- (a) True
 - (b) False
8. A binary search tree of n nodes has a worst-case depth of $O(n \log n)$
- (a) True
 - (b) False
9. Radix Sort ALWAYS runs in $O(n)$.
- (a) True
 - (b) False
10. Bubble sort is the slowest sorting algorithm
- (a) True
 - (b) False
11. In C++, the *protected* modifier for members of a class C means:
- (a) The exact same as private
 - (b) The same as private except when C inherits from another class
 - (c) The same as private except when another class derives from C
 - (d) There may be only 1 copy of the class in memory at a time
12. Suppose you have a pointer variable p and you execute the statement *delete p*;. What happens?
- (a) p is deleted from the heap.
 - (b) p is deleted from the stack.
 - (c) The variable which p points to is deleted but p 's value remains the same
 - (d) All variables that point to p are deleted, p remains unaffected.
13. If the definition of class Foo contains the line “friend class Bar;” then
- (a) Instances of type Foo can access “private” members of instances of type Bar
 - (b) Instances of type Bar can access “private” members of instances of type Foo
 - (c) Instances of type Bar can access “friend” members of instances of type Foo
 - (d) None of the above

14. Removing an item from an arbitrary position in an array-based list takes worst-case
- (a) $O(1)$
 - (b) $O(\log n)$
 - (c) $O(n)$
 - (d) $O(n \log n)$
15. Suppose you have a **sorted** array of 250 elements. What is the maximum number of elements you have to examine to determine if some value v is in your array if you use an efficient search?
- (a) 7
 - (b) 8
 - (c) 125
 - (d) 250
16. Which is the correct **increasing**¹ ordering of the following runtime complexities:
 $O(n), O(2^n), O(n^3), O(n \log n), O(n^2), O(\log n), O(n!)$
- (a) $O(\log n), O(n), O(n \log n), O(n^2), O(n^3), O(2^n), O(n!)$
 - (b) $O(\log n), O(n \log n), O(n), O(n^2), O(2^n), O(n^3), O(n!)$
 - (c) $O(n!), O(\log n), O(n \log n), O(n), O(n^2), O(2^n), O(n^3)$
 - (d) $O(\log n), O(n \log n), O(n), O(n^2), O(n^3), O(n!), O(2^n)$
17. Which of the following data structures would be most appropriate for representing (graphically) the organization of the US Government?
- (a) A list
 - (b) A stack
 - (c) A queue
 - (d) A tree

¹slowest-growing to fastest-growing

18. Which of the following data structures would be most appropriate for representing path of a traveler who will take the reverse path on the return trip?
- (a) A list
 - (b) A stack
 - (c) A queue
 - (d) A tree
19. Which of the following is the run-time for the *dequeue* operation on a linked-list-based queue?
- (a) $O(1)$
 - (b) $O(\log n)$
 - (c) $O(n)$
 - (d) $O(n \log n)$
 - (e) $O(n^2)$
20. Which of the following is the average-case run-time for the *insert* operation on a BST?
- (a) $O(1)$
 - (b) $O(\log n)$
 - (c) $O(n)$
 - (d) $O(n \log n)$
 - (e) $O(n^2)$
21. Which of the following is the run-time for QuickSort if the largest value in the list is always chosen as the pivot?
- (a) $O(1)$
 - (b) $O(\log n)$
 - (c) $O(n)$
 - (d) $O(n \log n)$
 - (e) $O(n^2)$

22. Which of the following is the run-time for Binary Search on a sorted array?

- (a) $O(1)$
- (b) $O(\log n)$
- (c) $O(n)$
- (d) $O(n \log n)$
- (e) $O(n^2)$

23. What is the running time of the following code

```
void f(int n)
{
  for (int i = 0; i < n; i++)
  {
    for (int j = 1; j < n; j *= 2)
    {
      // Something O(1)
    }
  }

  for (int i = 0; i < 3 * n; i++)
  {
    // Something O(1)
  }
}
```

- (a) $O(1)$
- (b) $O(\log n)$
- (c) $O(n)$
- (d) $O(n \log n)$
- (e) $O(n^2)$

24. If the above function f takes 8 seconds to run on an input of size k , about how long will it take on an input of size $2k$?
- (a) 2 seconds
 - (b) 8 seconds
 - (c) 10 seconds
 - (d) 18 seconds
 - (e) Not enough information
25. What is *NOT* wrong with the following code?

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

- (a) Bad recursive step
- (b) Probable segfault
- (c) Doesn't print in order
- (d) Doesn't compile
- (e) None of the above

2 Short Answer

26. Write out the class/struct definition for a doubly-linked list of strings.

```
struct Node
{
    string val;
    Node* prev;
    Node* next;
};
```

27. Assuming you have a function `void merge(vector<int>& nums, int begin, int end)` which merges two sorted lists together (one whose indices are $begin \dots (begin + end)/2 - 1$, and one whose indices are $(begin + end)/2 \dots end - 1$), write `void mergeSort(vector<int>& nums, int begin, int end)`.

```
void mergeSort(vector<int>& nums, int begin, int end)
{
    if (begin + 1 >= end) return;
    int mid = (begin + end) / 2;
    mergeSort(nums, begin, mid);
    mergeSort(nums, mid, end);
    merge(nums, begin, end);
}
```

28. Correct the following code (there are at least 10 errors)

```
struct Node* // Should be no *
{
private: // Should not be private
    int val;
    Node left; // Should be Node*
    Node right; // Should be Node*
};

Node* insert(Node* root, int val)
{
    if (root == NULL)
    {
        Node* root = new Node; // Should not be declaring a new var, use root
        root->val = val;
        root->left = NULL;
        root->right = NULL;
        return NULL; // Should return root instead
    }
    // Add if (val < root->val)
    root->left = insert(root->left, val);
    // Add else
    root->right = insert(root->right, val);

    return NULL; // Return root, not NULL
}

void printTree(Node* root)
{
    if (root != NULL) return; // if root == NULL
    printTree(root->right); // Probably should be left then right
    cout << root->val << " ";
    printTree(root->left);
}

int main()
{
    Node root = NULL;
    insert(root, 10); // Ignoring the return value! root = insert(root, 10)
    insert(root, 15); // root =
    insert(root, 5); // root =
    cout << root << endl; // call printTree?
}
```

29. Write *enqueue* for a queue implemented on a singly-linked list with a tail pointer.

```
// MUST be done at tail, otherwise we get O(n) dequeue on a singly
// linked list!
void Queue::enqueue(int val)
{
    if (head != NULL)
    {
        tail->next = new Node;
        tail = tail->next;
        tail->val = val;
    } else {
        head = new Node;
        head->val = val;
        tail = head;
    }
}
```

30. Write out C++ code for the $O(n^2)$ sorting algorithm of your choice.

```
void gnomeSort(vector<int>& nums)
{
    unsigned int pos = 1;
    unsigned int s = nums.size();
    while (pos < s)
    {
        if (pos == 0 or nums[pos] >= nums[pos - 1])
        {
            pos++;
        } else {
            swap(nums[pos], nums[pos - 1]);
            pos--;
        }
    }
}
```

31. Write *bool hasVal(Node* root, int val)* to find whether the value *val* is found in the BST rooted at *root*. You may write it iteratively or recursively.

```
bool hasVal(Node* root, int val)
{
    if (root == NULL) return false;
    if (val < root->val) return hasVal(root->left, val);
    if (val > root->val) return hasVal(root->right, val);
    return true;
}
```

32. Write a function in C++ that takes in an integer *n* and runs in time $O((\log n)^2)$. It doesn't need to do anything interesting.

```
void f(int n)
{
    for (int i = 1; i < n; i *= 2)
    {
        for (int j = 1; j < n; j *= 2)
        {
            // Something O(1)
        }
    }
}
```

33. Describe a situation where the use of exceptions would be helpful.

Exceptions are useful in unexpected or “exceptional” situations. Good examples include when performing bounds-checking on arrays/lists, removing a value from an empty heap, and taking the square root of a negative value.

34. Describe a situation where the use of templates would be helpful.

Templates are very useful in any situation where data structures or functions operating on different types could be copy-and-pasted with only the types changing. Data structures are particularly appropriate candidates for templating, since the behavior of most data structures doesn't change based on the type they are storing.