

CS 14 - Summer 2003 - Midterm Examination

July 23, 2003

1 Multiple Choice

Circle the letter of the correct answer.

1. 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) $\lceil \log_2 250 \rceil = 7.96$
 - (c) 125
 - (d) 250
2. What is the worst-case behavior for Quick-sort?
 - (a) $O(\lg n)$
 - (b) $O(n)$
 - (c) $O(n \lg n)$
 - (d) $O(n^2)$
3. What is the benefit of using a linked-list over an array-based list?
 - (a) No preset size limit
 - (b) No “shifting” to move data around on insertion or removal, which is good when copying is expensive
 - (c) **Both a and b**
 - (d) None of the above
4. Which is the correct **increasing** 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) $O(\log n), O(n), O(n \log n), O(n^2), O(n^3), O(2^n), O(n!) \leftarrow$
 - (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) $O(\log n), O(n \log n), O(n), O(n^2), O(n^3), O(n!), O(2^n)$
5. Removing an item from an arbitrary position in an array-based list takes worst-case
- (a) $O(1)$
 - (b) $O(\lg n)$
 - (c) $O(n) \leftarrow$
 - (d) $O(n \lg n)$
6. Removing an item from the end of an array-based list takes worst-case
- (a) $O(1) \leftarrow$
 - (b) $O(\lg n)$
 - (c) $O(n)$
 - (d) $O(n^2)$
7. If a binary tree does **not** explicitly keep track of its depth, what is the best Big-O running time of the function that calculates the depth of the tree?
- (a) $O(1)$
 - (b) $O(\lg n)$
 - (c) $O(n) \leftarrow$
 - (d) $O(n \lg n)$
8. An algorithm that is $O(n^2)$ is also $O(n^3)$
- (a) **True**
 - (b) False
9. 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
10. A 2-3 Tree is a type of
- (a) Binary search tree
 - (b) **Search tree**
 - (c) Binary tree
 - (d) AVL tree
11. How many data items can a 2-3 Tree can contain in a leaf node?

- (a) 1
 - (b) **1 or 2**
 - (c) 2
 - (d) 2 or 3
12. A 2-3 Tree storing n items will generally have a smaller depth than a binary search tree storing n items, but will require about the same number of comparisons to traverse from root to leaf.
- (a) **True**
 - (b) False, it will require fewer comparisons
 - (c) False, it will require more comparisons
 - (d) There is no way to compare the two structures
13. The *partition* step of Quick-sort can be done without allocating additional storage in $O(\lg n)$ steps
- (a) True
 - (b) **False, it requires $O(n)$ steps**
 - (c) False, it cannot be done in place but does take $O(\lg n)$ steps
 - (d) False, it cannot be done in place and requires $O(n)$ steps
14. 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.
15. If you have a sorted list of n elements, it is always most efficient to use binary-search to find a given element.
- (a) True
 - (b) **False** (You may be using a linked list, in which case lookup is $O(n)$!)
16. 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
17. An algorithm that is $O(n^3)$ is also $O(n^2)$

- (a) True
 - (b) **False**
18. It is best for a linked-list based Stack to add and remove from the **tail** of the list.
- (a) True
 - (b) **False**
19. An iterator makes retrieval of each object from a container (in a pre-set order) take $O(1)$ per object
- (a) **True**
 - (b) False
20. All recursive algorithms must have a base case
- (a) **True**
 - (b) False
21. A heap is a binary tree that has either the minimum or maximum element stored in the root, among other properties.
- (a) **True**
 - (b) False
22. An in-order traversal of a heap prints the elements in sorted order.
- (a) True
 - (b) **False**
23. An iterator can only be used with a list data-type; it can never be used with any other container.
- (a) True
 - (b) **False**
24. A Red-Black tree is an example of
- (a) A basic binary tree
 - (b) A 2-3 Tree
 - (c) **A 2-3-4 Tree**
 - (d) None of the above
25. Which of the following data structures would be most appropriate for representing the organization of the US Government?
- (a) A list
 - (b) A stack

- (c) A queue
 - (d) **A tree** - Rooted at the Constitution!
26. 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

2 Short Answer

If you want the option to request a regrade available, your answers to these questions must be made in pen.

1. Describe the condition required for the worst-case behavior of Quick-sort to be occur. If the pivots that are chosen create partitions of the form $n - m$ and m on input size of n . For example, if the pivot is always the max element, then the partitions are $n - 1$ and 1 . This leads to n^2 behavior because each partitioning is linear, and there are now a linear number of partitions that must be created.
2. Write the function `int findMax(Node* n)` for a singly-linked list.

```
int findMax(Node* n)
{
  int max = n->val;
  n = n->next;
  while (n != NULL)
  {
    if (max < n->val) max = n->val;
    n = n->next;
  }
  return max;
}
```

3. Write the function `Node* insert(Node* root, int val)` for a binary search tree. Recall that it needs to return the (possibly new) root Node and allocate memory for the new Node. Begin by writing the case for inserting into an empty tree.

```
Node* insert(Node* root, int val)
{
  if (root == NULL)
```

```

{
Node* newNode = new Node;
newNode->val = val;
return newNode;
}
if (val < root->val) root->left = insert(root->left, val);
if (val > root->val) root->right = insert(root->right, val);
return root;
}

```

4. Given the function *int partition(vector<int>& nums, int low, int high)* which picks a pivot, partitions the appropriate range of *nums* around that pivot, and returns the index of the pivot, write

```

void quickSort(vector<int>& nums, int low, int high)
void quickSort(vector<int>& nums, int low, int high)
{
if (low == high) return;
int pivot = partition(nums, low, high);
quickSort(nums, lo, pivot - 1);
quickSort(nums, pivot + 1, high);
}

```

5. Write *int minIndex(int a[], int length)*, a C++ function that returns the index of the minimum element in the array *a*, which has length *length*.

```

int minIndex(int a[], int length)
{
int minInd = 0;
for (int i = 1; i < length; i++)
{
if (a[i] < a[minInd]) minInd = i;
}
return minInd;
}

```

6. Is it useful to define an Abstract Data Type, like ADT List or ADT Stack, without explicitly defining its implementation (array, linked-list, etc)? Explain why or why not.
Yes, this way we ensure encapsulation / modularity of our code, increase code reusability, allow for separate development of sections of a project, and uphold the Object Oriented paradigm.
7. Draw the binary search tree formed by starting with an empty tree and adding the values 8, 4, 10, 3, 11, 12, 13, 9 in that order.
See <http://www.cs.ucr.edu/cs14-p/cs14.03sum/midtermtree.png>
8. What is the **in-order** traversal of the binary search tree described above (Hint: think about what an in-order traversal is.)
3, 4, 8, 9, 10, 11, 12, 13
9. Describe in English the algorithm for Merge-sort.
Recursively split the list into two halves, stopping when the lists are of size 1 and thus sorted. Each step in the recursive stack now merges the (sorted) results of its recursive calls in $O(n)$ using a temporary array.

10. What is the worst-case Big-O running time of a function that finds the median¹ element of an unsorted list? Explain in English what the algorithm would be and why it has that running time.

If the list were sorted, this would take either $O(n)$ or $O(1)$ depending on the implementation of the *size* method for the list. In either case, it would be dwarfed by the running time of the *sorting* call required to make the list sorted. Without having the list sorted, we cannot efficiently find the median. Thus, the Big-O would be the running time of our sorting algorithm.

11. **Bonus Credit** If you have two lists of integers in the range 1..100, describe a method for determining if they are actually the same list (although possibly reordered).

Allocate an array of size 100. Run through the first list and count the number of times each value occurs using this array (thus if we find a value of 42 we increment the 42nd value in the array). Repeat with the second list, only decreasing the values this time. Finally, run through the array we allocated and make sure everything in it is 0. If anything is non-zero, return false. Otherwise, return true.

¹median: middle element in a sorted list