EECS 281, Week 4: Wednesday

Asymptotics

1. Fill in the following table by selecting all relations ρ such that $f(x)$ is $\rho(g(x))$.

Question	$f(x)$	$g(x)$	f(x) is … of g(x)		
			O	Ω	Θ
Example	$203x^2$	$203x$		✔	
1.1	x	$x + 183$	✔		✔
1.2	280^x	281^x	✔		
1.3	$\log x$	$\sqrt{x}$	✔		

2. Order from most to least efficient:

   $O(n)$ $O(n!)$ $O(n^n)$ $O(n^2)$ $O(2^n)$ $O(\log n)$ $O(1)$
   $O\left(\sqrt{n}\right)$ $O(n^3)$ $O(n^2 \log n)$ $O(3^n)$ $O(n \log n)$

   $O(1) \subset O(\log n) \subset O\left(\sqrt{n}\right) \subset O(n) \subset O(n \log n) \subset O(n^2) \subset O(n^2 \log n) \subset$
   $O(n^3) \subset O(2^n) \subset O(3^n) \subset O(n!) \subset O(n^n)$.

3. (T/F) If $f_1(n) = O(g_1(n))$ and $f_2(m) = O(g_2(m))$, then $f_1(n) \cdot f_2(m) = O(g_1(n) \cdot g_2(m))$.

   True

4. (T/F) It is possible for an $\Omega(n^3)$ algorithm to grow at a rate slower than an $\Omega(n)$ algorithm.

   True

5. If the running time of an (improved search) algorithm is $f(n) = \log n$, so that the time it takes for an input of size S is $T = \log S$, then in time $T + 1$ you can solve a problem of size _____. *(Express your answer in terms of S.)*

   Answer: $2S$. Each time the input S doubles, the time it takes to solve the problem becomes $\log(2S)$. By the logarithmic property $\log(m \cdot n) = \log m + \log n$, $\log(2S) = \log 2 + \log S$. Since we assume base 2, $\log 2 = 1$, and so $\log(2S) = 1 + \log S = T + 1$.

6. If the running time of an algorithm is $f(n) = n^2$, so that the time it takes for an input of size S is $T = S^2$, then in time $2T$ you can solve a problem of size _____. (Express your answer in terms of S.)

   Answer: $\sqrt{2}S$. Let S' be the size of problems that you can solve in $2T$ time, such that $2T = S'^2$. Then $S'^2 = 2 \cdot S^2$ and $S' = \sqrt{2}S$.

7. Several sorting algorithms have a running time of $f(n) = n \log n$, so that the time it takes for an input of size S is $T = S \log S$. If you instead need to sort <u>two</u> lists, each of size S, you will need time _____ to solve your new problem. (Express your answer in terms of T.)

   Answer: $2T$. Since it's actually two (unrelated) problems of the same size, and not a single problem twice as big, the total time is simply doubled.

8. Suppose we find that an algorithm has a runtime $R(N)$ that is quadratic in the worst case, and linear in the best case. Which of the following statements are true?

   A. $R(N) \in O(N)$ — False
   B. $R(N) \in O(N^2)$ — True
   C. $R(N) \in O(N^3)$ — True
   D. $R(N) \in \Theta(N)$ — False
   E. $R(N) \in \Theta(N^2)$ — False
   F. $R(N) \in \Theta(N^3)$ — False
   G. In the worst case, $R(N) \in O(N)$ — False
   H. In the worst case, $R(N) \in O(N^2)$ — True
   I. In the worst case, $R(N) \in O(N^3)$ — True
   J. In the worst case, $R(N) \in \Theta(N)$ — False
   K. In the worst case, $R(N) \in \Theta(N^2)$ — True
   L. In the worst case, $R(N) \in \Theta(N^3)$ — False
   M. In the best case, $R(N) \in O(N)$ — True
   N. In the best case, $R(N) \in O(N^2)$ — True
   O. In the best case, $R(N) \in O(N^3)$ — True
   P. In the best case, $R(N) \in \Theta(N)$ — True
   Q. In the best case, $R(N) \in \Theta(N^2)$ — False
   R. In the best case, $R(N) \in \Theta(N^3)$ — False

9. $1 + 2 + 3 + 4 + \ldots + N =$

   Answer: $\frac{N(N+1)}{2} \in \Theta(N^2)$

10. $1 + 2 + 4 + 8 + 16 + \ldots + N =$

    Answer: $2N - 1 \in \Theta(n)$

Analysis of Algorithms

11. Find a simple $f(n)$ such that the runtime $R(n) \in \Theta(f(n))$.

```
int fn(int n) {
    if (n <= 1) {
        return 1;
    }
    return fn(n / 2) + fn(n / 2);
}
```

Answer: N

EECS 281, Week 4: Wednesday

12. Find a simple $f(n)$ such that the runtime $R(n) \in \Theta(f(n))$.

```
int recursive(int n) {
    if (n <= 1) {
        return 1;
    }
    return recursive(n - 1) + recursive(n - 1);
}
```

Answer: 2^N. Each iteration spawns two iterations. Thus, by the time we get to the bottom level (where $n = 1$), we've produced 2^n total calls to fn.

13. Find a simple $f(n)$ such that the runtime $R(n) \in \Theta(f(n))$.

```
int fn(int n) {
    int x = 0;
    for (int i = 0; i < n; i++) {
        x++;
    }
    return x;
}
```

Answer: N. This is the similar to the pattern in Merge sort, except that each recursive call does only a constant amount of work instead of a linear amount. At the top level, we do 1 unit of work; at the 2nd level, we do 2 units of work; at the 3rd level, we do 4 units, etc. The total amount of work is thus given by $1 + 2 + 4 + 8 + \ldots + N$. This sum is linear in N.

14. Find a simple $f(n)$ such that the runtime $R(n) \in \Theta(f(n))$.

```
int fn(int n) {
    int x = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < i * i; j++) {
            x++;
        }
    }
    return x;
}
```

Answer: N^3 Each iteration of the inner loop is quadratic in the outer loop variable. The simplest way to do this is to realize we're just summing $\sum i^2$, which will just be $O(N^3)$.

15. Find a simple $f(n)$ such that the runtime $R(n) \in \Theta(f(n))$. Assume that `fn2` takes linear time.

```
int fn(int n) {
    if (n <= 1) {
        return 1;
    }
    return fn2(n) + fn(n / 2) + fn(n / 2);
}
```

Answer: $N \log N$. This is the same pattern as Merge sort. If you want to think of it as a sum, then it's $N + N + \ldots + N$, repeated $\log_2 N$ times.

16. Describe the difference between best-case, worst-case, average-case and amortized complexity.

EECS 281, Week 4: Wednesday

Recurrence Relations

If possible, use the Master Theorem to find the complexity class for the following recurrence relations. Express your answers using Big Θ notation. If the Master Theorem does not apply, enter "N/A" and explain why not.

17. $T(n) = 3T\left(\frac{n}{4}\right) + n^2$

18. $T(n) = nT\left(\frac{n}{2}\right)$

19. $T(n) = 8T\left(\frac{n}{2}\right) + n^3$

20. $T(n) = 2T(n-1)$

21. $T(n) = 4T\left(\frac{n}{2}\right)$

EECS 281, Week 4: Wednesday

STL

22. If there are A algorithms and C containers, STL must provide only _____ total number of implementation.

    $A + C$

23. Why are there several kinds of iterators?

    Answer: Not all data structures are built the same, and they can't all support every kind of motion. For example, a linked list cannot be accessed in a random order.

24. What is the difference between iterators and output iterators?

    Answer TODO

25. (T/F) STL list only provides forward iterators.

    False

26. (T/F) Bidirectional iterators can be compared with operators < and >.

    False

27. (T/F) For a valid STL container `c`, when the expression `c.end() - c.begin()` is well-defined, it returns the same value as `c.size()`.

    True

28. Given an STL container called `aContainer`, what is the difference between `aContainer.begin()`, `aContainer.end()`, `aContainer.cbegin()`, `aContainer.cend()`, `aContainer.rbegin()`, `aContainer.rend()`, `aContainer.crbegin()` and `aContainer.crend()`?

    Answer TODO

29. (T/F) An iterator always holds an address (pointer), and operator++ applied to the iterator always increases that address.

    False

30. Which STL containers among deque, vector and list can invalidate iterators, pointers and references, and when?

    vector: iterators, pointers and references are invalidated after insertion if new container size is greater than the previous capacity.

    deque: iterators, pointers and references are invalidated after insertion (unless inserting at the front/end, then references are unaffected).

    list: iterators, pointers and references are unaffected.

31. (T/F) For a valid STL container `aContainer.rend`, the iterator returned by `aContainer.rend()` refers to the first valid element in `aContainer`.

    False

32. What are the drawbacks of using an STL vector?

    expensive reallocation, requires contiguous memory

33. How to minimize reallocations of a vector?

    if number of elements is known, use reserve function

    use container.shrink_to_fit() to trim off unused memory.

34. How is an STL deque different from an STL vector?

    Deque grows linearly, unlike vector that grows exponentially. No reallocations. Deque has no reserve() and no capacity() functions. Slightly slower than vector. More complex data structure. Data is not in contiguous memory, problem of locality (more page faults, cache misses).

35. Describe the situations when a deque is preferred over a vector and when a vector is preferred over a deque.

    If need to push_front a lot, use deque. If performance is important, use vector. in general, vector is faster than a deque.

    Element type: when the elements are not of primitive type, deque is not much less efficient than vector. The cost of the construction and the destruction starts to dominate the performance and the performance of the containers themselves becomes less important.

    Memory Availability: vector requires data to reside in contiguous memory chunk. Prefer using deque if allocation of large contiguous memory is a problem.

    Frequency of growth. For vector, reallocation is expensive. Workaround: use reserve(). Prefer deque if growth is unpredictable.

    Also consider invalidation of pointers/references/iterators because of growth. Deque does a better job at maintaining the integrity of pointers

    Vector can be used with functions that work with arrays: &v[0] can be used as a raw pointer and v.size() is the size.

Containers, Linked Lists, Arrays

36. What is the difference between an ordered container and a sorted container?

    An ordered container maintains elements in a specific order. The order is independent of the value. A sorted container maintains elements in a specific order that depends on the values of the elements.

    Queue is an example of a container that is ordered but not sorted.

37. (T/F) A sorted container must be ordered.

    True

38. (T/F) Assuming the list has no tail pointer, appending to the end of the list is O(n).

    True

39. A linked list is normally specified by giving a pointer pointing to the first node of a list. An empty list has no nodes at all. What is the value of a head pointer for an empty list?

    Null pointer.

40. (T/F) If the data that a link list contains is small, it generally take up less memory than an STL vector of the same length.

    False

41. (T/F) For both linked lists and arrays, it takes linear time to insert something if you want to maintain it in sorted order.

    True. For different reasons.

42. Show that doing a binary search on a sorted linked list takes $\Theta(n)$ time.

    Answer TODO

43. (T/F) Inserting an element at a random index in a vector has tightest upper bound $O(n)$.

    True

44. (T/F) Inserting an element at the end of a vector has tightest upper bound $O(n)$.

    True

45. In a _____ array with n elements, kth element can be removed in worst-case $O(1)$ time, but in a _____ array with n elements, this may take $O(n)$.

    not sorted; sorted

46. Describe at least three situations when linked lists are preferred over arrays.

    Fast insertion/removal at the beginning/end, splicing, contiguous memory is a problem

47. Describe at least three situations when arrays are preferred over linked lists.

    Random access, binary search, faster bc of locality (fewer cache misses)

Queues and Stacks

48. (T/F) In a queue, items are added and removed according to the LIFO principle.

    False

49. In a stack, items are added and removed according to the FIFO principle.

    False

50. How can we implement a queue that can hold at most N elements using an array of size N?

    Answer: TODO

51. Describe how to implement a queue with two stacks.

    See `http://maximal.io/eecs281/4m/src4m/newQueue.cpp`

EECS 281, Week 4: Wednesday

Priority Queues and Heaps

52. (T/F) The height of a binary heap with n elements is $O(n)$.

    True, although not the tightest bound, which is $\Theta(\log n)$.

53. If we represent a binary heap with a zero-indexed array, what is the index of the parent of the element at index i?

    $\frac{i-1}{2}$

54. What is the index of the parent of the left child of the element at index i?

    $2i + 1$

55. What is the index of the parent of the right child of the element at index i?

    $2i + 2$

56. What is the height of the tree with n elements?

    $\lceil \log(n+1) \rceil$

57. (T/F) buidMaxHeap is an algorithm that can turn any array into a heap in $\Theta(n)$ time, where n is the number of elements in the array.

    True

58. Which of the following represents a max-heap?

    A. [47, 8, 12, 9, 2, 10, 10, 4, 3, 1]

    B. [2, 13, 8, 16, 13, 10, 40, 25, 17]

    C. [3, 5, 6, 7, 12, 15, 14, 9, 10, 11]

    D. [59, 58, 60, 57, 85, 49, 32, 21, 5]

    E. None of the above.

    Answer: E. None of the above.

59. Suppose that seven values are pushed into an empty maximum priority queue binary heap in the following order:

    9, 7, 6, 17, 21, 1, 14

    What would be the array representation of the heap after all seven values are inserted? What would be the tree representation?

60. What would the array representation of the heap from the previous question look like after two deletions? Tree representation?

61. Suppose that seven values are pushed into an empty minimum priority queue binary heap in the following order:

    12, 4, 9, 27, 13, 2, 6

    What would be the array representation of the heap after all seven values are inserted? What would be the tree representation?

62. What would the array representation of the heap from the previous question look like after two deletions? Tree representation?

63. Suppose we wish to create a binary heap containing the keys `H E L L O W O R L D`. (All comparisons use alphabetical order.) Show the resulting min-heap if we build it using successive `insert()` operations (starting from `H`).

64. Suppose we wish to create a binary heap containing the keys `H E L L O W O R L D`. (All comparisons use alphabetical order.) Show the resulting min-heap if we build it using `buildMinHeap()`.

Disjoint sets

65. (T/F) Representing the items in a disjoint set as a tree and storing the true identity of the set at the root provides a fast find operation and a slower union operation.

    False

66. If we restrict the items in the universe to N integers from 0 to 1, how can we represent tree-based disjoint sets as an array?

    See `https://datastructures.maximal.io/disjoint-sets/`

67. What optimization can we make to the union operation on tree-based disjoint sets?

    See `https://datastructures.maximal.io/disjoint-sets/`

68. What optimization can we make to the find operation on tree-based disjoint sets?

    See `https://datastructures.maximal.io/disjoint-sets/`

69. Write the find function for a tree-based disjoint set represented as an array that leverages the optimization from the previous question.

```
void find(int x) {
    if (array[x] < 0) {
        return x;
    } else {
        array[x] = find(array[x]);
        return array[x];
    }
}
```

Sorting

70. What is an inversion?

    A pair of elements that are out of order.

71. What is a stable sort?

    A sort that maintains the relative order of equal elements.

72. What is an in-place sort?

    A sort that uses a small, constant amount of additional memory.

73. What are the properties of sorting algorithms that we care about?

    Computational complexity: best case, average case, worst case. Comparison-based algorithms need at least O(n log n) comparisons on most inputs.

    Memory: A sort is in-place if it uses a small, constant amount of additional memory.

    Adaptiveness: does the presortedness of the input positively affects the running time?

    Stability: maintain the relative order of equal items.

74. (T/F) All in-place sorting algorithms are stable.

    False. For example, heapsort.

75. (T/F) Running time of insertion sort is proportional to the number of inversions.

    True

76. (T/F) Selection sort takes $\Theta(n)$ time in the best case and $\Theta(n^2)$ time in the worst case.

    False

77. (T/F) Merge sort is a divide-and-conquer algorithm and most of the work is done in the "dividing" part.

    False

78. Write a recurrence relation for merge sort.

    $T(0) = T(1) = 1$

    $T(n) = 2T\left(\frac{n}{2}\right) + n$

79. (T/F) Merge sort is suited well for linked lists.

    True

80. (T/F) In-place versions of merge sort exist and outperform quicksort.

    False

81. (T/F) Quicksort is a divide-and-conquer algorithm and most of the work is done in the "conquering" part.

    False

82. (T/F) Worst case time complexity of quicksort is $\Theta(n \log n)$.

    False

83. (T/F) Worst case time complexity of a randomized quicksort algorithm is $\Theta(n \log n)$.

    False

84. (T/F) Quicksort is suited well for linked lists.

    False

85. (T/F) Quicksort cannot be performed in-place.

    False

86. (T/F) The most optimal partitioning policy for quicksort on an array we know nothing about would be selecting a random element in the array.

    True

87. Write a recurrence relation for quicksort.

    $T(0) = T(1) = 1$

    $T(n) = T(n-1) + n$

88. State at least four ways to optimize sorting algorithms.

    Switch to insertion sort for small arrays.
    Make sorts adaptive: exploit existing order in array (insertion sort).
    Exploit restriction on set of keys: 3-way quicksort, Dutch national flag problem.
    Quicksort: make introspective. Switch to a different sort if recursion goes too deep.

89. (T/F) The fastest possible comparison sort has a worst case no better than O(n log n).

    True

90. Give an example of a situation where using insertion sort is more efficient than using merge sort.

    Answer: Insertion sort performs better than merge sort for lists that are already almost in sorted order (i.e. if the list has only a few elements out of place or if all elements are within k positions of their proper place and k < logN).

91. Match the sorting algorithms to the sequences, each of which represents several intermediate steps in the sorting of an array of integers.

    - Sort W

      ```
      12,  7,  8,  4, 10,  2,  5, 34, 14
       7,  8,  4, 10,  2,  5, 12, 34, 14
       4,  2,  5,  7,  8, 10, 12, 14, 34
      ```

    - Sort X

      ```
      23, 45, 12, 4, 65, 34, 20, 43
      12, 23, 45, 4, 65, 34, 20, 43
      ```

    - Sort Y

      ```
      45, 23,  5, 65, 34,  3, 76, 25
      23, 45,  5, 65,  3, 34, 25, 76
       5, 23, 45, 65,  3, 25, 34, 76
      ```

    - Sort Z

      ```
      23, 44, 12, 11, 54, 33, 1, 41
      54, 44, 33, 41, 23, 12, 1, 11
      44, 41, 33, 11, 23, 12, 1, 54
      ```

    W: Quicksort.

    X: Insertion sort.

    Y: Merge sort.

    Z: Heapsort.

92. Given an array containing the elements `[6, 1, 7, 0, 7, 3, 9, 5]`, show how the order of the elements changes during each step of…
    92.1. bubble sort
    92.2. insertion sort
    92.3. selection sort
    92.4. mergesort
    92.5. quicksort
    92.6. heapsort

93. Some sorting algorithms are not naturally stable. Suggest a way to make any sorting algorithm stable by extending the keys.

    Extend each item so that it has a "secondary key," which is the index of the item in the initial array/list. If two items have the same primary key, the tie is broken using the secondary key, so no two items are ever considered equal.

94. Suppose we do the following:
    - Read 1,000,000 integers from a file into an array of length 1,000,000.
    - Use merge sort to sort these integers.
    - Randomly select one integer and change it.
    - Sort using algorithm S of your choice.

    Which sorting algorithm would be the fastest choice for S?

    A. Selection sort
    B. Heap sort
    C. Merge sort
    D. Insertion sort

    Anser: D. Insertion sort.

95. Fill this table:

Algorithm	Best case	Average case	Worst case	Memory	Stable	Notes
Bubble sort						
Selection sort						
Insertion sort						
Heapsort						
Merge sort						
Quicksort						
Random Quicksort						

See `https://datastructures.maximal.io/sorting/summary/`

Miscellaneous

96. Define deep copy and shallow copy.

    Shallow copies duplicate as little as possible. A shallow copy of a collection is a copy of the collection structure, not the elements. With a shallow copy, two collections now share the individual elements.

    Deep copies duplicate everything. A deep copy of a collection is two collections with all of the elements in the original collection duplicated.

97. Define memory leak.

    A resource leak that occurs when a computer program incorrectly manages memory allocations in such a way that memory which is no longer needed is not released.

98. Explain the copy-swap method.

    Used for operator =. Create a copy of the object with a copy constructor. Swap member variables with the `std::swap` function. Return `*this`.

Additional Practice Questions

99. Write a function template for a function named minimum. The function will have two parameters of the same type. It returns the smaller of these (either if they are equal.)

100. Implement merge function using iterators.

```
template <class InputIterator1, class InputIterator2, class OutputIterator>
OutputIterator merge(InputIterator1 first1, InputIterator1 last1,
                     InputIterator2 first2, InputIterator2 last2,
                     OutputIterator result) {


    // while there are elements in left and right side
    while (first1 != last1 && first2 != last2) {
        // compare elements in the beginning of each side
        if (*first1 < *first2) {
            // copy element from left side
            *result = *first1;
            ++first1;
        }
        else {
            // copy element from right size
            *result = *first2;
            ++first2;
        }
        ++result;
    }

    // check if exhausted left side
    if (first1 == last1) {
        // copy elements from right side
        while (first2 != last2) {
            *result++ = *first2++;
        }
    } else {
        // copy element from left side
        while (first1 != last1)
        {
            *result++ = *first1++;
        }
    }
    // return iterator pointing to the past-the-end element in the result
    return result;


}
```

101. Given a function that partitions an array on a pivot in linear time using constant space (you don't need to worry about the pivot selection or the partitioning method), implement a function to find median using partitioning.

```
int partition(double a[], int left, int right);
```

```
int findMedian(int a[], int leftIndex, rightIndex) {



keep partitioning until the pivot ends up as the middle element
after each partition, only partition the larger half (where the middle index
would be)

}
```

102. What is the time complexity of finding the median using partitioning?

$\Theta(N \log N)$ in the worst case

103. Find if a linked list has a loop/cycle in O(n) time.

104. Reverse a singly-linked list.

105. Implement `isPermutation`, a function that takes two vectors and returns true if one is a permutation of the other and false otherwise. In other words, the function should return true if the two vectors contain the same elements (although possibly in different order) and false otherwise. Time complexity should be better than $\Theta(n)$.

```
bool isPermutation(const vector<int> &v1, const vector<int> &v2) {



check size. if different, return false
make copies of both vectors
sort both copies
compare element by element

}
```