DAA Midterm, Fall 2017

Professor Callahan

Name: _____

NetID: _____

Multiple Choice

2 points each

1. Consider a max heap, represented by the array [16, 14, 9, 10, 7, 8, 3, 1, 4, 2].
The value 12 is inserted into this heap. After heap_insert(), the new array is:
a. [16, 14, 9, 10, 12, 8, 3, 1, 4, 2, 7]
b. [16, 12, 9, 10, 14, 8, 3, 1, 4, 2, 7]
c. [16, 14, 9, 4, 12, 8, 3, 1, 10, 2, 7]
d. [16, 14, 3, 10, 12, 8, 9, 1, 4, 2, 7]

Answer: a. [16, 14, 9, 10, 12, 8, 3, 1, 4, 2, 7]

2. According to the master method, the recurrence $T(n) = 4T(n / 2) + n^2 \lg n$ is
a. Big-Theta(n)
b. Big-Theta(n^2)
c. Big-Theta($n \lg n$)
d. The master method is not applicable

Answer: d. The master method is not applicable

3. When multiplying an m * n matrix with an n * p matrix, the straightforward run-time complexity (without using something like Strassen's algorithm) is:
a. m * p
b. n * p
c. m * n * p
d. m * lg n * p

Answer: c

4. What is the Big-O of 3T(n/3) + 12n^3?
a. $O(n^4)$
b. $O(n^3)$
c. O(n!)
d. $O(n^{1/3})$

Answer: b. $O(n^3)$
Solving the recurrence by Masters Theorem

5. Huffman's algorithm is a greedy algorithm because:
a. Several frequencies are combined at each step.
b. We always look ahead to future probability choices when making the current choice.
c. It involves drawing a Huffman tree.
d. We never look ahead to future probability choices when making the current choice.

Answer: d

6. Strassen's algorithm speeds up the asymptotic complexity of matrix multiplication, but at the cost of:
a. higher setup cost
b. worse behavior as inputs get large
c. greater code complexity
d. a and c

Answer: d

7. If a graph contains several minimal spanning trees (MSTs):
a. they will all have the same number of edges
b. each MST will have a unique number of edges
c. the number of edges in each MST will be equal to the number of vertices in the graph
d. the number of edges will be calculated from the graph's matroid

Answer: a

8. What is the time complexity for the following piece of code?

```
sum = 0;
for (int i = 0; i < n; i = i + 1)
   for (j = 1; j < n; j = j + 1)
      sum += n;
```

a. $O(n^2)$

b. $O(n)$
c. $O(\lg n)$
d. $O(n \lg n)$
Answer: a

Reason : The outer loop will run with time complexity of n and inner loop will run with time complexity of n. Hence, $O(n^2)$.

10. Recurrences can be handled by the master theorem:

a. So long as the coefficient of the recurrence (a in $aT(n/b)$) is less than one.
b. So long as divisor of the amount of work (b above) is less than one.
c. So long as there is a less-than-polynomial difference between the recurrence portion and the work portion.
d. When none of the above are true.

Answer: d, as a, b, and c are the conditions when we can't apply the master theorem.

11. A common alternative to bottom-up dynamic programming, which performs roughly as well, is:
a. a binary search tree
b. recursion with memoization
c. a greedy algorithm
d. an exponential algorithm

Answer: b.
Reason: c could perform better, d worse, and a is just irrelevant.

12. In a vector space, matroid theory tells us that:
a. every basis will consist of the same number of vectors
b. every independent subset of the vector space will be of the same size
c. we can form a basis of the vector space from however many vectors we please
d. one basis for a vector space may be much larger than another such basis

Answer: a

13. A thief enters a store and sees the following items: Item A worth $120 weighing 2 pounds, Item B worth $20 weighing 2 pounds and Item C worth $150 weighing 3

pounds. His Knapsack holds 4 pounds. What is his maximum profit according to the Fractional Knapsack Problem?
a. 220
b. 195
c. 180
d. None of the above

Answer: a. 220

Explanation: In the Fractional Knapsack problem, the item with the maximum profit by weight is chosen first. In this case the maximum profit is obtained by choosing Item A first(120/2 is equal to 60).
Profit: 120
Item A weighs 2 pounds so there are 2 more pounds remaining in the knapsack. The next maximum 'by weight' profit is Item C (150/3 is equal to 50). So we take 2 pounds of Item C.
Final profit: (2*60) + (2*50) = 220

14. Dynamic programming is a good choice when we have:
a. optimal subproblems
b. overlapping subproblems
c. a situation in which the greedy choice won't work
d. all of the above

Answer: d

15. In dynamic programming, the technique of storing previously calculated values is called:
a. Hashing
b. Memoization
c. Saving the value properly
d. Top-down programming

Answer: b. Memoization
Reason: It memorizes or stores the results for subproblems.

16. If $f(n) = 3n^4 + n^3 \lg n$, then $f(n)$ is
a. $O(n^4)$
b. $O(n^{3/2})$

c. O(n^3 lg n)
d. O(n^2/3)

Answer: a

Reason : Highest power of n is n^4

17. According to the master theorem ,the recurrence $T(n) = T(n - 1) + T(n - 3)$ is:
a. not solvable with the master theorem
b. O(n^2)
c. O(n lg n)
d. O(n - 1)

Answer: a

18. Greedy algorithms are widely used in memory management. Given a list of blocks, in which the sizes are { 20, 120, 140, 240, 340 }, how would a greedy algorithm (grab the smallest block that works first) fit processes with sizes { 10, 200, 108, 137, 308 }?
a. process 1: block 1; process 2: block 2; process 3: block 4; process 4: block 3; process 5: block 5
b. process 1: block 1; process 2: block 5; process 3: block 4; process 4: block 3; process 5: block 2
c. process 1: block 5; process 2: block 2; process 3: block 4; process 4: block 3; process 5: block 1
d. process 1: block 1; process 2: block 4; process 3: block 2; process 4: block 3; process 5: block 5

Answer: d

19. If $f(x) = (x^8 - 1) / (5x^4 + 1)$ then f(x) is (choose tightest bound):
a. O(x^4)
b. O(x^8)
c. O(x^8/5x)
d. O(1)

Answer: a

Reason : The highest power of x will be x^4.

20. What is big-O for following code?

```
f(n):
    for i from 1 to log(n):
        for j from 1 to n:
            print(j)
```

a. O(n)
b. O(n lg n)
c. O(n^2)
d. None of the above

Answer: b

Reason : As we can see the inner loop will run n times and the outer loop runs for lg n times so lg n times n operations would be executed so complexity is n lg n.

21. A greedy algorithm can be used to solve dynamic programming problems when:
a. There are overlapping subproblems.
b. The cost of a choice is locally contained.
c. The cost of a choice affects other parts of the problem.
d. All of the above.

Answer: b

Reason: A greedy algorithm gives optimal solution for all subproblems, but when these locally optimal solutions are combined it may NOT result into a globally optimal solution. Hence, a greedy algorithm CANNOT be used to solve all the dynamic programming problems.

22. Which of the following is true about Huffman coding.
a. Huffman coding may become lossy in some cases
b. Huffman codes rely on a linked list data structure
c. In Huffman coding, no code is the prefix of any other code.
d. All of the above

Answer: c

Reason: Huffman coding is a lossless data compression algorithm. The codes

assigned to input characters are Prefix Codes, means the codes are assigned in such a way that the code assigned to one character is not prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding.

23. An example of an abstract data type (ADT) is:
a. a linked list
b. an array
c. a 64-bit integer
d. an iterator

Answer: d. (That is the best answer. Some computer scientists say a and b are correct as wel.)

24. What is big O for following code?

```
f(n)
{
    for(i = 1; i < n; i = i*2)
    {
        print("Algorithms")
    }
}
```

a. O(n)
b. O(lg n)
c. O(n lg n)
d. O(n^2)

Answer: b

Reason: As we can see after each iteration of the loop the value of i increase by 2. As we know that Time Complexity of a loop is considered as O(lg n) if the loop variable is multiplied by a constant amount greater than 1.

25. The time complexity of heap sort in worst case is

a) O(lg n)
b) O(n)
c) O(n lg n)
d) O(n^2)

Answer: c

26. The following sequence is a Fibonacci sequence:
0, 1, 1, 2, 3, 5, 8, 13…
Which method can be used to get the nth Fibonacci term?

a) Dynamic Programming
b) Recursion
c) An iterative solution
d) All of the above

Answer: d.

27. Consider a rod of length 5 and the prices for each length:

Length	Price
1	3
2	6
3	7
4	8
5	8

What combination of its pieces would give you the maximum price?

a) {1, 4}
b) {1, 2, 2}
c) {2, 3}
d) {5}

Answer:b {1,2,2} which gives a max price of 15

28. You are looking at a series of investments that can be made each January. However, financial regulations require you to hold an investment for five years once your are in it, during which time all of your money will be locked up in that investment. Given you have a list, r, of the revenue you expect from each investment, what is the recurrence relation you should use to pick out the maximum revenue you can expect from the entire series?
a. T(*n*) = r[*n*] + T(*n* - 5)
b. T(*n*) = max(r[*n*], T(*n* - 5))
c. T(*n*) = max(r[*n*] + T(*n* - 5), T(*n* - 1))
d. T(*n*) = max(r[*n*] + T(*n* - 1), T(*n* - 5))

Answer: c. T(n) = max(r[n] + T(n - 5), T(n - 1))

29. According to the master method, the recurrence $T(n) = 4T(n / 2) + n^2$ is
a. Big-Theta(n^2)
b. Big-Theta($n^2 \lg n$)
c. Big-Theta($n \lg n$)
d. The master method is not applicable

Answer: a. n^2

30. Which of the following is a property of a dynamic programming problem?
a. Binary substructure
b. No overlapping subproblems
c. Greedy approach
d. None of the above

Answer: d

31. We have one algorithm for processing customer records with runtime of O(n), and another with runtime of O(lg n) + 2000. In what circumstances might we want to choose the O(n) algorithm?
a. No circumstances.
b. If our programmers are really bad.
c. We believe our program will always be dealing with a number of records less than 2000.
d. If n is very large.

Answer: c

32. Θ-notation applied to a function implies it is
a. a function we know little about.
b. asymptotically bound from above and below.
c. asymptotically bound from below only.
d. asymptotically bound from above only.

Answer: b

33. Besides running time, we can also measure algorithm performance by:
a. disk usage
b. memory usage
c. power consumed
d. all of the above

Answer: d

34. In algorithm analysis, we usually analyze algorithms in terms of
a. actual running time in nanoseconds.
b. the number of disk operations.
c. the number of basic operations.
d. CPU time used.

Answer: c

35. In algorithm analysis, we usually look at
a. worst-case performance.
b. average-case performance.
c. best-case performance.
d. median-case performance.

Answer: a

36. Ω-notation applied to a function implies
a. it is a function we know little about.
b. it is asymptotically bounded from above and below.
c. only that it is asymptotically bounded from below.
d. only that it is asymptotically bounded from above.

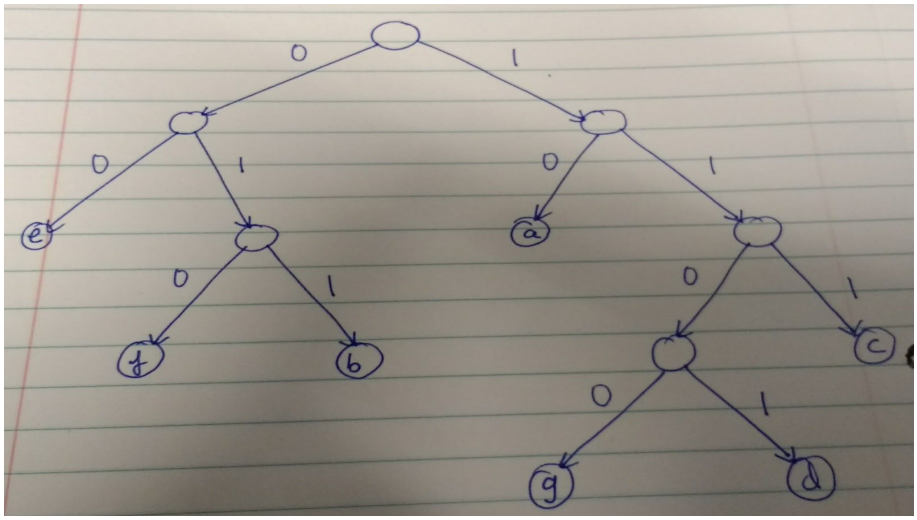
Answer: c

Problems

4 points each

1. Draw the Huffman encoding that will deal with the following letters and their corresponding frequencies:
(g, 6), (d,13), (f,17), (b,18), (c, 29), (e, 30), (a, 37)

Answer:



(It's not necessary to draw a tree although it is preferable)

a -> 10
b -> 011
c -> 111
d -> 1101
e -> 00
f -> 010
g -> 1100

2. Draw the recurrence tree for the recurrence T(n) = T(n / 3) + T(2n / 3) + n. Show the amount of work at each level.

Answer: https://math.stackexchange.com/questions/1112012/recursion-tree-tn-tn-3-t2n-3-cn

3. Walk through the dynamic programming matrix chain multiplication algorithm and write down what the algorithm puts in the result array each step of the way if it is fed as input matrices 15x5, 5x10, 10x20, and 20x25. (Recall CLRS would represent this in their pseudo-code as 15, 5, 10, 20, 25.)
HINT: Your answer should look something like this (although obviously these aren't the right numbers!):

```
                15000
        2000            4000
1000            2500            1000
```

Answer:

```
                5375
        2500            3500
750             1000            5000
```

4. Consider the recurrence $T(n) = T(n - 2) + T(n - 4)$, with base cases:
$T(0) = 0$
$T(1) = 7$
$T(2) = 0$
$T(3) = 7$

Please write memoized pseudocode to solve this recurrence for an arbitrary $n > 3$ efficiently.

Answer:

Something like this:

```
def memo_weird(n, first_call=True):
    results = []
    if first_call:
```

```
    # any similar initialization will do!
    results = [-1 for x in range(1000)]
  if n < 0:
    print("n must be >= 0!")
    return -1
  elif n == 0:
    return 0
  elif n == 1:
    return 7
  elif n == 2:
    return 0
  elif n == 3:
    return 7
  else:
    # Here we check and see if we have calculated the results
    # we need earlier. If so, use them. If not, calculate and
    # then store ("memo-ize") them.
    n_minus2 = 0
    n_minus4 = 0
    if results[n - 2] >= 0:
      n_minus2 = results[n - 2]
    else:
      n_minus2 = memo_weird(n - 2, False)
      results[n - 2] = n_minus2
    if results[n - 4] >= 0:
      n_minus4 = results[n - 4]
    else:
      n_minus4 = memo_weird(n - 4, False)
      results[n - 4] = n_minus4
    return n_minus2 + n_minus4
```

6. Trace through a (max) heapsort of [25, 24, 21, 20, 13, 17, 16, 12]. Show the array after build-heap and after each max-heapify from the heapsort loop. (You do NOT need to show the heapifies within build-heap.)

Answer:

After build_heap, h = [25, 24, 21, 20, 13, 17, 16, 12]
Looping in heapsort with i = 7; h = [25, 24, 21, 20, 13, 17, 16, 12]
Looping in heapsort with i = 6; h = [24, 20, 21, 12, 13, 17, 16, 25]
Looping in heapsort with i = 5; h = [21, 20, 17, 12, 13, 16, 24, 25]
Looping in heapsort with i = 4; h = [20, 16, 17, 12, 13, 21, 24, 25]
Looping in heapsort with i = 3; h = [17, 16, 13, 12, 20, 21, 24, 25]

Looping in heapsort with i = 2; h = [16, 12, 13, 17, 20, 21, 24, 25]
Looping in heapsort with i = 1; h = [13, 12, 16, 17, 20, 21, 24, 25]
Looping in heapsort with i = 0; h = [12, 13, 16, 17, 20, 21, 24, 25]

7. You are running a 100K ultra-marathon from Brooklyn to Connecticut. There are frequent water stops (supplied for your information in array *w*) along the way. You want to stop at as few of them as possible (because stopping slows you down) but without fainting from dehydration. You determine you can run *k* kilometers without fainting. Write a greedy algorithm that will guide you to stop at the fewest water stops possible without fainting. Explain why no algorithm can do better than yours.

Answer:

```
S = empty_set
last = 0
for i = 1 to number_water_stops:
        if(w[i] - last) > k:
                S union w[i - 1]
                last = w[i - 1]
```

"Can't do better": that's a cut-and-paste proof: since we took the last water stop we could at each step, any set choosing an earlier stop could at best tie us for fewest stops.