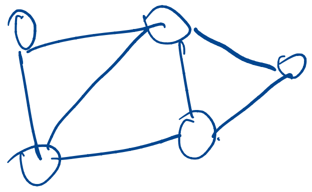
CSCI-570 Fall 2023

Practice Midterm 1

[redacted]

INSTRUCTIONS

- The duration of the exam is 140 minutes, closed book and notes.
- No space other than the pages on the exam booklet will be scanned for grading! Do not write your solutions on the back of the pages.
- If you require an additional page for a question, you can use the extra page provided within this booklet. However please indicate clearly that you are continuing the solution on the additional page.



1. True/False Questions

a) (T/[✓ F]) Dynamic Programming approach only works on problems with non-overlapping sub problems.

b) In a dynamic programming solution, the space requirement is always at least as big as the number of unique sub problems. F

c) (T/[✓ F]) To determine whether two binary search trees on the same set of keys have identical tree structures, one could perform an inorder tree walk on both and compare the output lists. F

d) (T/[✓ F]) If graph G has more than $V-1$ edges, and there is a unique heaviest edge, then this edge cannot be part of a minimum spanning tree.

e) ([✓ T]/F) If the lightest edge in a graph is unique, then it must be part of every MST.

f) (T/[✓ F]) If $f(n) = \Omega(n \log n)$ and $g(n) = O(n^2 \log n)$, then $f(n) = O(g(n))$.

g) ([✓ T]/F) The shortest path in a weighted directed acyclic graph can be found in linear time.

h) (T/[✓ F]) The Standard Merge sort will take $\theta(n^2)$ time in the worst case.

i) ([✓ T]/F) The recurrence equation T(n)=$2 * T(\frac{n}{2}) + 3n$ has the solution $T(n) = \theta(\log(n^2) * n)$

j) ([✓ T]/F) Suppose that there is an algorithm that merges two sorted arrays in $\sqrt{n}$ then the merge sort algorithm runs in $O(n)$ $2T(n/2) + n^{0.5}$

k) (T/[✓ F]) The memory space required for any dynamic programming algorithm with n^2 unique subproblems is $\Omega(n^2)$

$$c_1 n\log n \le f(n)^2 \le c_2 n^2 \log n$$

l) (T [✓ F]) In a 0/1 knapsack problem with n items, suppose the value of the optimal solution for all unique subproblems has been found. If one adds a new item to the list now, one must re-compute all values of the optimal solutions for all unique subproblems in order to find the value of the optimal solution for the n+1 items.

m) (T [✓ F]) In dynamic programming you must calculate the optimal value of a sub-problem twice, once during the bottom up pass and once during the top down pass.

2. Multiple Choice Questions

a) Consider the recurrence relation for two algorithms given as:

1) $T_1(n) = 3 * T_1(\frac{n}{2}) + n^2$ $\theta(n^2)$

2) $T_2(n) = 2 * T_2(\frac{n}{2}) + n * logn$

Select the option which represents the correct Asymptotic time complexity for $T_1(n)$ and $T_2(n)$ respectively.

[✓ a) $\theta(n^2)$, $\theta(n * log^2 n)$]

b) $\theta(n)$, $\theta(n * log^2 n)$

c) $\theta(n^2)$, $\theta(n^2 * logn)$

d) None of the above

b) If a binomial heap contains these three trees in the root list: B_0, B_1, and B_3, after 2 DeletetMin operations it will have the following trees in the root list.

[✓ a) B_0 and B_3]

b) B_1 and B_2

c) B_0, B_1 and B_2

d) None of the above

c) Consider a complete graph G with 4 vertices. How many spanning tree does graph G have?

a) 15
b) 8
[✓ c) 16]
d) 13

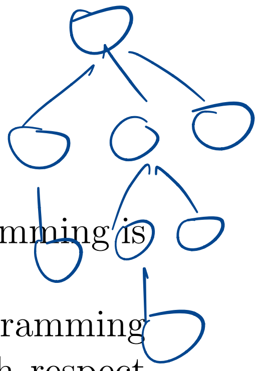
$n^{n-2} = 4^{4-2} = 16$

d) Which of the following is false about Prim's algorithm?

a) It is a greedy algorithm.
[✓ b) It constructs MST by selecting edges in increasing order of their weights.]
c) It never accepts cycles in the MST.
d) It can be implemented using the Fibonacci heap.

e) The solution to the recurrence relation $T(n) = 8T(n/4)+O(n^{1.5}\log n)$ by the Master theorem is

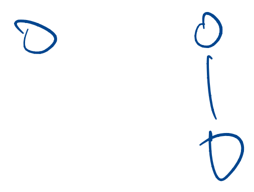
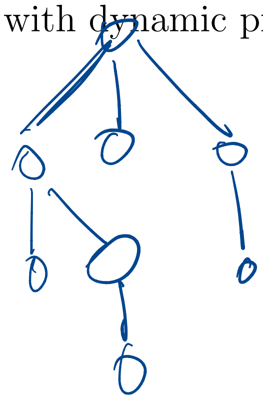
a) $O(n^2)$
b) $O(n^2 \log n)$
c) $O(n^{1.5} \log n)$
[✓ d) $O(n^{1.5} \log^2 n)$]

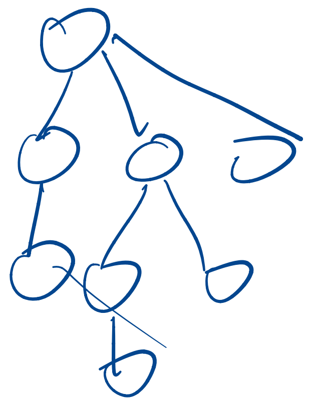


f) Which of the following statement about dynamic programming is correct?

No right answer

a) Any problem that can be solved using dynamic programming has a polynomial time worst case time complexity with respect to its input size.
b) Dynamic Programming approach only works on problems with non-overlapping subproblems.
c) 0/1 knapsack problem can be solved using dynamic programming in polynomial time.
d) In the sequence alignment problem, the optimal solution can be found in linear time by incorporating the divide-and-conquer technique with dynamic programming.



g) Which description about dynamic programming is correct?

a) Dynamic programming is exclusively used for sorting algorithms.

b) Dynamic programming can only be applied to problems with linear complexity.

[✓ c)] Dynamic programming relies on solving problems by dividing them into smaller, overlapping subproblems.

d) Dynamic programming always guarantees the fastest possible runtime for any algorithm.

3. Dynamic Programming Algorithm

USC students get a lot of free food at various events. Suppose you have a schedule of the next n days marked with those days when you get a free dinner, and those days on which you must acquire dinner on your own. On any given day you can buy dinner at the EVK Dining Hall for \$7 . Alternatively, you can purchase one week's groceries for \$21, which will provide dinner for each day that week. However, as you don't own a fridge, the groceries will go bad after seven days and any leftovers must be discarded. Due to your very busy schedule (midterms), these are your only three options for dinner each night.

Write a dynamic programming algorithm to determine, given the schedule of free meals, the minimum amount of money you must spend to make sure you have dinner each night. The iterative version of your algorithm must have a polynomial run-time in n.

a) Define (in plain English) sub-problems to be solved.

b) Write a recurrence relation for the sub-problems

c) Using the recurrence formula in part b, write an iterative pseudo-code to find the solution.

d) Make sure you specify

- base cases and their values
- where the final answer can be found

e) What is the complexity of your solution?

$$\min(i) : \min\begin{pmatrix} \min(i-1) + 7 \\ \min(i-7) + 21 \text{ if } i > 7 \\ 0 \text{ if meal is free on } i \end{pmatrix}$$

4. Divide and Conquer Algorithm

You are given a sorted array of n positive integers such that $A[1] < A[2] < \ldots < A[n]$. Your job is to determine if there exists an array index k, such that $k = A[k]$.

Give a detailed divide and conquer algorithm for the Boolean function that returns FALSE if no such k can be found, and returns TRUE if such an index exists. The complexity of this function must be $O(\log n)$. Prove that your algorithm will also run in $O(\log n)$.

```
def find_fixed_point_recursive(arr, left, right):
    if left > right:
        return -1  # No fixed point found

    mid = left + (right - left) // 2

    if arr[mid] == mid:
        return mid  # Found a fixed point at index mid
    elif arr[mid] > mid:
        # If arr[mid] is greater than mid, the fixed point must be on the left side
        return find_fixed_point_recursive(arr, left, mid - 1)
    else:
        # If arr[mid] is less than mid, the fixed point must be on the right side
        return find_fixed_point_recursive(arr, mid + 1, right)

# Example usage:
arr = [-10, -5, 2, 2, 2, 3, 4, 7, 9, 12, 13]
result = find_fixed_point_recursive(arr, 0, len(arr) - 1)

if result != -1:
    print(f"Fixed point found at index {result}")
else:
    print("No fixed point found")
```

5. Heaps

Design a data structure that has the following properties:

- Find median takes $O(1)$ time.
- Extract-Median takes $O(\log n)$ time.
- Insert takes $O(\log n)$ time.
- Delete takes $O(\log n)$ time.

where n is the number of elements in your data structure. Describe how your data structure will work and provide algorithms for all aforementioned operations. You are not required to prove the correctness of your algorithm.

1 2 3 4 5 6 7 8 9 10

Insert : With an incoming element
- if max heap is empty push to maxheap
- else if element is lesser than root of max heap push to maxheap
- else push to min heap
- else if len(maxheap) − len(minheap) ≥ 2 delete from maxheap & push to minheap
- else the vice versa of above point

Delete :
- remove the root from the given heap
- maintain ordering property by percolating

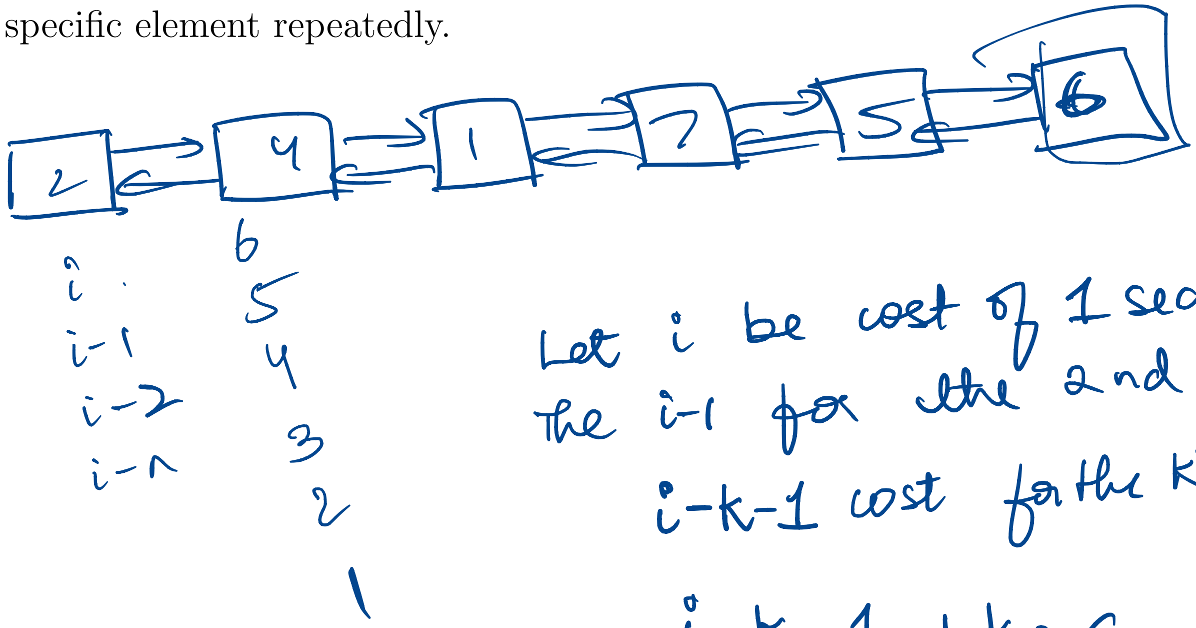
Extract Median :
- if len(maxheap) > len(minheap)
  - Delete the root of maxheap
  - return it
  - restructure
- if len(minheap) > len(maxheap)
  - Delete the root of minheap
  - return it
  - restructure
- if len(max) == len(min)
  - Delete root of both max & min
  - return max+min/2

6. **Greedy Algorithm**

   You have a finite set of points $P = \{p_1, p_2, p_3, \ldots, p_n\}$ along a real line. Your task is to find the smallest number of intervals, each of length 2, which would contain all the given points. For example, when $P = \{5.7,\ 8.8,\ 9.1,\ 1.5,\ 2.0,\ 2.1,\ 10.2\}$, an optimal solution has 3 intervals $[1.5, 3.5]$, $[4, 6]$, and $[8.5, 10.5]$ are length-2 intervals such that every element is contained in one of the intervals. Device a greedy algorithm for the above problem and analyze its time complexity. You don't need to prove the correctness of your algorithm.

7. Amortized Analysis

In a data structure, there is a doubly linked list with 'head' and 'tail' pointers for storing n elements. When searching for a specific element, the search starts from the head and proceeds until the target element is found, at an index i where in. The cost for each 'search' operation is i, and when the target element is found, it's moved towards the head of the list by 1 index at a cost of c (a positive constant). We want to calculate the amortized time complexity for searching and moving an element in the worst-case scenario. Assume that the number of times the target element is searched is less than or equal to the total number of elements in the list, and consider a scenario where we search for this specific element repeatedly.



Let i be cost of 1 search

The $i-1$ for the 2nd search

$i-k-1$ cost for the kth search

$i-k-1+k\cdot c$

moving

$+k\cdot c$

$i+n-1+n-2 \cdots +$

Total cost $= \sum_{n=0}^{k=1} n+n-1+\cdots 1 + n\cdot c$

$= \frac{n(n+1)}{2} + n\cdot c = \frac{n^2+n+2nc}{2}$

$= O(n^2)$

$= ~~O(n^2)~~$ $= O(n)$