Anash Mehta
USCID: 6498-8280-04

CSCI 570 - HW3

Graded Problems

1. We have N ropes having lengths $L_1, L_2 ..., L_N$. We can connect two ropes at a time: Connecting ropes of length L and L' gives a single rope of length $L+L'$ and doing so has a cost of $L+L'$. We want to repeatedly perform such connections to finally obtain one single rope from the given N ropes. Develop an algorithm to do so, while minimizing the total cost of connecting. No proof is required. (10 points)

   Algorithm:

   Make a min-heap of the length of the ropes

   Repeat the process until there is no child of the root element

   Extract the first two min heap elements

   Add the length of the ropes

   Push the sum in the heap

   Re-heapify the min-heap

   End loop

   Return the top element of the min-heap

2. There are N tasks that need to be completed using 2 computers A and B. Each task i has 2 parts that take time: a_i (first part) and b_i (second part) to be completed. The first part must be completed before starting the second part. Computer A does the first part of all the tasks while computer B does the second part of all the tasks. Computer A can only do one task at a time, while computer B can do any number of tasks at the same time. Find an $O(n \log n)$ algorithm that minimizes the time to complete all the tasks and give a proof of why the solution obtained by the algorithm is optimal. (15 points)

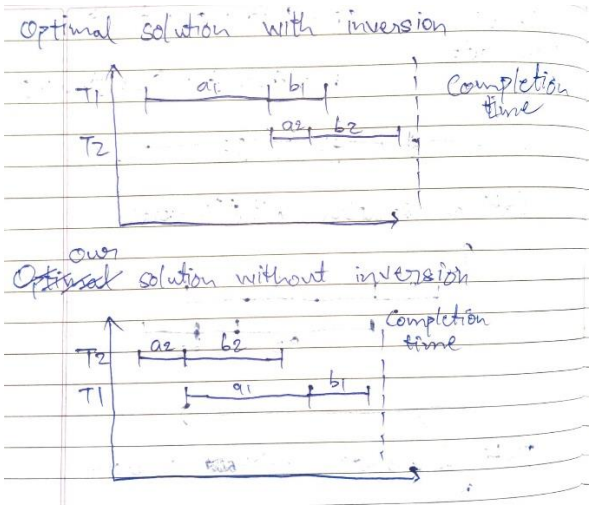
   Answer:

   Since computer A can perform only one task at a time, the time at which computer A ends its processing of the tasks does not matter – it will always be a constant i.e., sum of first part of all the tasks. Hence, the algorithm must focus on ordering of the second task since it can handle multiple tasks at the same time.

   Proof of Correctness:

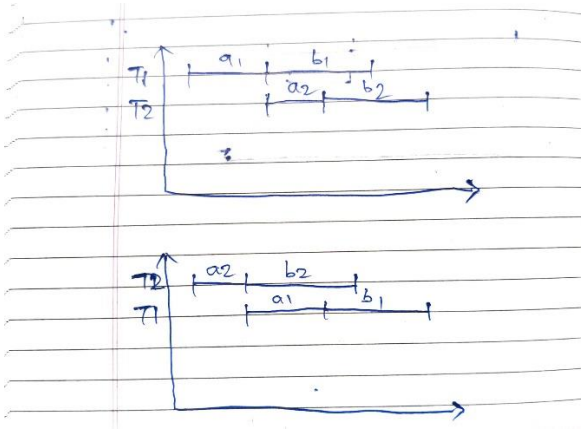
   We can prove our solution is optimal by changing an optimal solution such that we do not lose its optimality.

   (i) Inversion is a property in which a task Ti with second task time bi is scheduled earlier to a task Tj with second task time bj where $bi < bj$.
   We assume an optimal solution with inversions and by removing these inversions, we can match it with our solution.
   For instance, we can observe it in the below picture, where there is an optimal solution with inversions and our solution without inversions:



Here, the optimal solution represents an inversion, and by swapping the tasks, we see that we an earlier completion time is observed – hence, we safeguarded the optimality.

(ii) If there are two solutions with duplicate *bi*, the same completion time can be observed. We see it in the below example:



Here, since *b1 = b2*, we observe the same completion time. Hence, the order of the first part of tasks do not matter if the second part of tasks are of the same time.

(iii) There exist all solutions with no inversions that have the same completion time.
This statement holds true if:

(A) Second part of tasks are equal. We proved that in (ii) that all solutions will have the same completion time.

(B) Second part of tasks are not equal. Here, only one solution could exist – one in decreasing order of *bi.*

Hence, if we eliminate inversions from the optimal solution, we know that it does not affect the optimality as proved in (i). In (iii), it says that all the solutions with no inversions have the same completion time, and if we remove inversions from the optimal solution, the optimal solution and our solution will hence have the same completion time. Thus, our solution is the optimal solution.

Algorithm:

Order the jobs in a decreasing order of the second part of the task b_i

While all tasks are complete

Allocate a_i to computer A

Wait for computer A to complete a_i

Allocate b_i to computer B

End while

Time complexity:

The algorithm will run on a time complexity of $O(n \log n)$ since the sorting algorithm takes $O(n \log n)$ time, and the task allocation algorithm takes $O(n)$ time.

3. Suppose you want to drive from USC to Santa Monica. Your gas tank, when full, holds enough gas to go p miles. Suppose there are n gas stations along the route at distances $d_1 \leq d_2 \leq ... \leq d_n$ from USC. Assume that the distance between any neighbouring gas stations, and the distance between USC and the first gas station, as well as the distance between the last gas station and Santa Monica, are all at most p miles. Assume you start from USC with the tank full. Your goal is to make as few gas stops as possible along the way. Give the most efficient algorithm to determine which gas stations you should stop at and prove that your algorithm yields an optimal solution (i.e., the minimum number of gas stops). Give the time complexity of your algorithm as a function of n. (15 points)

Answer:

Suppose the gas stations are represented by the variable g. Here, the greedy strategy to stop for the least gas stations will be to go to stations that are less than or equal to p miles. Hence, we can visit as less gas stations as possible.

Algorithm:

Let A be an array which contains the distances between the gas stations

Let f be the distance travelled and D be the total distance from USC to Santa Monica

While $f + p < D$

Select farthest gas station that is less than p

Update $f = f + a_i$

End while

Proof of Correctness:

To prove the correctness of the algorithm, we will try matching our solution so that our solution 'stays ahead' of the optimal solution.

For the first gas station to refill, both the optimal solution and our solution will select the farthest gas station. Hence, for the first case, both solutions are equal.

Assume our solution's gas stations j and $j+1$ distance to be d_j and d_{j+1}

Assume optimal solutions' gas stations j and $j+1$ distance to be d'_j and d'_{j+1}

Also, assume that our solution's gas station k is farther away than optimal solution's gas station k.

There are 2 equations:

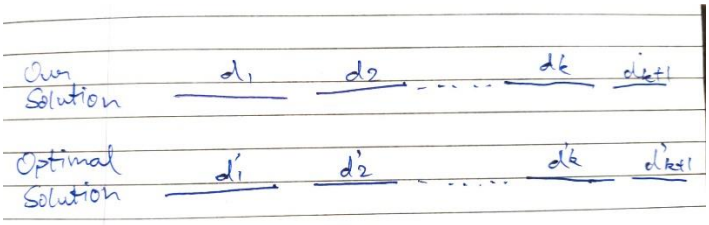
(i) $d_{j+1} \leq d_j + p$ and $d'_{j+1} \leq d'_j + p$
(ii) $d'_j \leq d_j$

Hence, it is imperative that $d'_{j+1} \leq d_{j+1}$

Therefore, we know that our solution and the optimal solution has equal number of gas stations, since a gas station cannot exist after d_{j+1}.

Thus, our solution is optimal.

We can explain it with an image below as well:



Time complexity: The loop runs for $O(n)$ time since it iterates once through the loop.

4. (a) Consider the problem of making change for n cents using the fewest number of coins. Describe a greedy algorithm to make change consisting of quarters (25 cents), dimes (10 cents), nickels (5 cents) and pennies (1 cents). Prove that your algorithm yields an optimal solution. (Hints: consider how many pennies, nickels, dimes and dime plus nickels are taken by an optimal solution at most.)
(b) For the previous problem, give a set of coin denominations for which the greedy algorithm does not yield an optimal solution. Assume that each coin's value is an integer. Your set should include a penny so that there is a solution for every value of n.

(a) Answer:

For a greedy approach, we will begin selecting the largest denomination first so that the change gets reduced the fastest.

Proof of Correctness:

Any optimal solution will try to optimize the n number of coins in the following way:

a) Pennies <= 4
   If there are pennies greater than 4, they can be combined to become one nickel.
b) Nickels <= 1
   If there are nickels greater than 1, they can be combined to become one dime.
c) Dimes + Nickels <= 2
   If there are 2 dimes and 1 nickel, they can be combined to become one quarter.

Thus, the optimal solution will try to reduce the change as much as possible, and this is what our greedy algorithm follows. Hence, our algorithm provides the optimal solution.

Algorithm:

Create an array *A* of size 4 which contains the number of each coin used for the change. Initialize them with 0.

Let *c* be the change you require

While *c > 0*

Find the largest denomination *d* where it is smaller than or equal to *c*

Update *c = c – d*

Increment the selected denomination in the array *A*

End while

Time complexity: This algorithm has one loop – so will run for *O(n)* time.

(b) Coin denominations = {1, 5, 15, 20, 25}. Suppose that we want a change of 35 cents. Our greedy algorithm will suggest us 1*25 and 2*5 = 35 i.e., 3 coins. Optimally though, the ideal solution will be of 2 coins (1*20 and 1*15 = 35 i.e., 2 coins).

5. Suppose you are given two sets *A* and *B*, each containing *n* positive integers. You can choose to order the numbers in each set however you like. After you order them, let a_i be the i^{th} number in set *A*, and let b_i be the i^{th} element of set *B*. You then receive a payoff of $\prod_{i=0}^{n} a_i^{b_i}$ Give an algorithm to decide the ordering of the numbers so to maximize your resultant payoff (6 points). Prove that your algorithm maximizes the payoff (10 points) and state its running time (2 points).

Answer:

If we be greedy, we want to pick the greatest elements from the sets *A* and *B*.

Proof of Correctness:

We prove this with the help of mathematical induction, where our solution always stays ahead of the optimal solution.

For the base case, we will be selecting the *max* (x_0, y_0) where x_0 and y_0 are the first elements from the list. So, either way, our solution will match the optimal solution.

For the next case, we will assume that our solution's *max* (x_i, y_i) is greater than the optimal solution's *max* (x_k, y_k). If we go by that assumption, hence, *max* (x_{i+1}, y_{i+1}) in our solution will be greater than *max* (x_{i+1}, y_{i+1}) in the optimal solution. Thus, the algorithm provided by us maximizes the payoff.

Algorithm:

Build two max-heaps of the lists A and B and name them A' and B' respectively

Let *t* be the payoff and initialize with 1.

While the heaps are not empty:

Extract max from *A'* and *B'*. Assign them to variables *x* and *y*

Update $t = t * (max (x^y, y^x))$

End while

Time complexity: The run time of the algorithm will be *O (n log n)*

6. The United States Commission of Southern California Universities (USCSCU) is researching the impact of class rank on student performance. For this research, they want to find a list of students ordered by GPA containing every student in California. However, each school only has an ordered list of its own students by GPA and the commission needs an algorithm to combine all the lists. Find the fastest algorithm for yielding the combined list and give its runtime in terms of *m*, the total number of students across all colleges, and *n*, the number of colleges. (12 points)

   Answer: We will build a min heap of the *n* lists provided by the universities. The greedy approach will try to merge the two smallest lists.

   Algorithm:
   Assume each node of the min-heap to contain the length of the list and a pointer to reference that list
   While nodes are greater than 1 in the min-heap:
   Extract the two nodes with the minimum sizes from the min-heap
   Perform a merge, build a new node for the min-heap and push it back into the min-heap
   End while
   The remaining node will be the combined node that will contain *m* entries

   Complexity:
   Let size of every list be m/n
   Here, a two-way merge is performed (n-1) times
   Thus,

   $$2m/n + 3m/n + 4m/n + \ldots. [(n-1) m]/n = O (mn)$$

   The Extract-Min and Insert operations of the min-heap would also me performed (n-1) times. So, the order would be O *(n log m).*

   Thus, the complexity of the algorithm is *O(mn)*

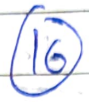
7. The array *A* below holds a max heap. What will be the order of elements in array *A* after a new entry with value 19 is inserted into this heap? Show all your work. *A* = {16,14,10,8,7,9,3,2,4,1}
   Answer in next page

Build a Max-Heap

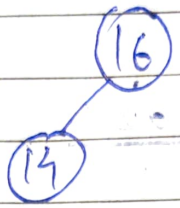
PAGE No.
DATE / /

Q7 $A = \{16, 14, 10, 8, 7, 9, 3, 2, 4, 1\}$

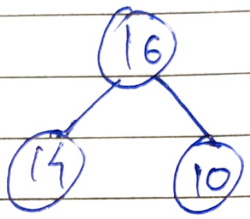
Add (16)



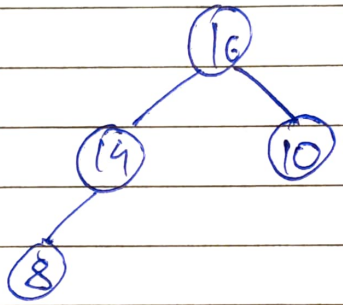
Add (14)



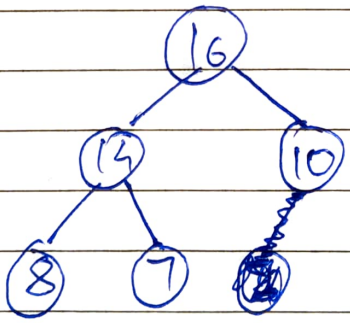
Add (10)



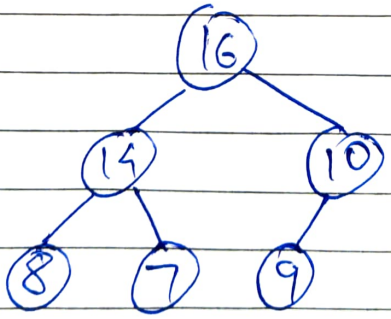
Add (8)



Add (7)

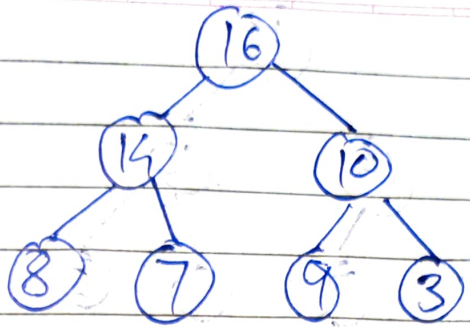


Add (9)

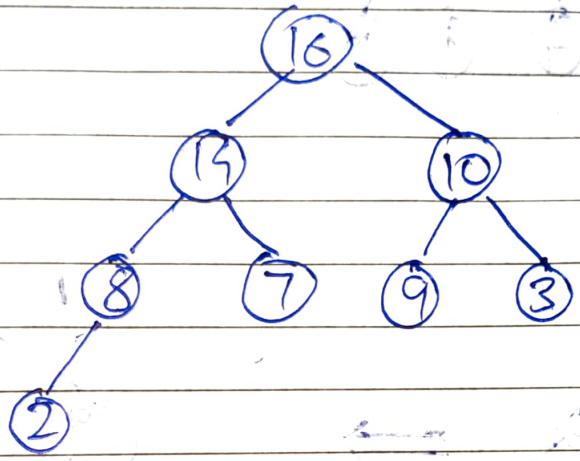


PAGE No.
DATE / /

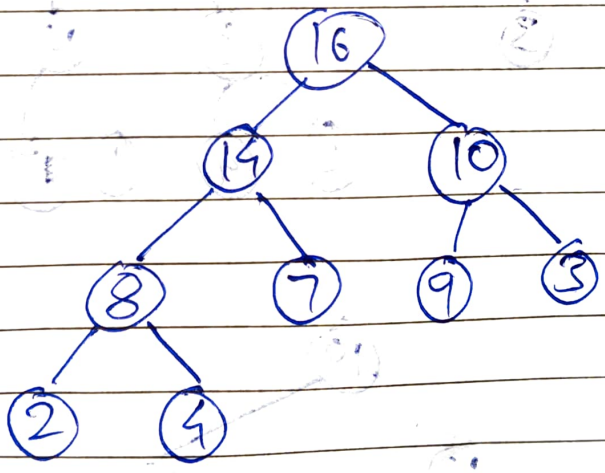
Add (3)



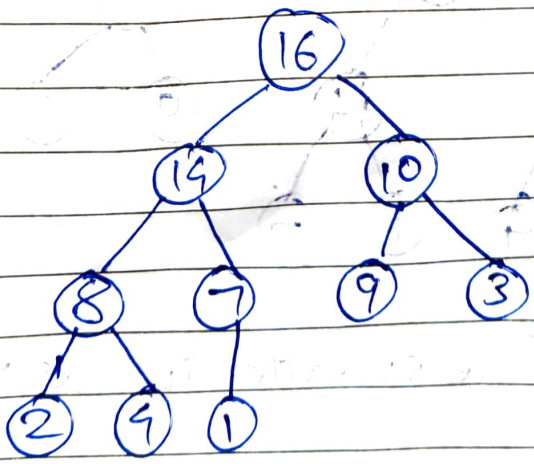
Add (2)



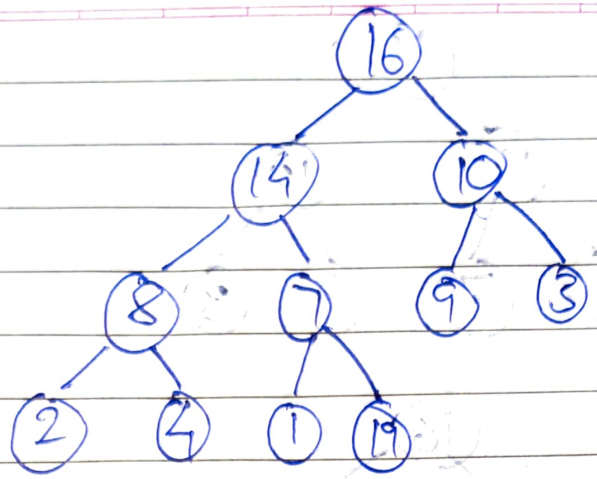
Add (4)



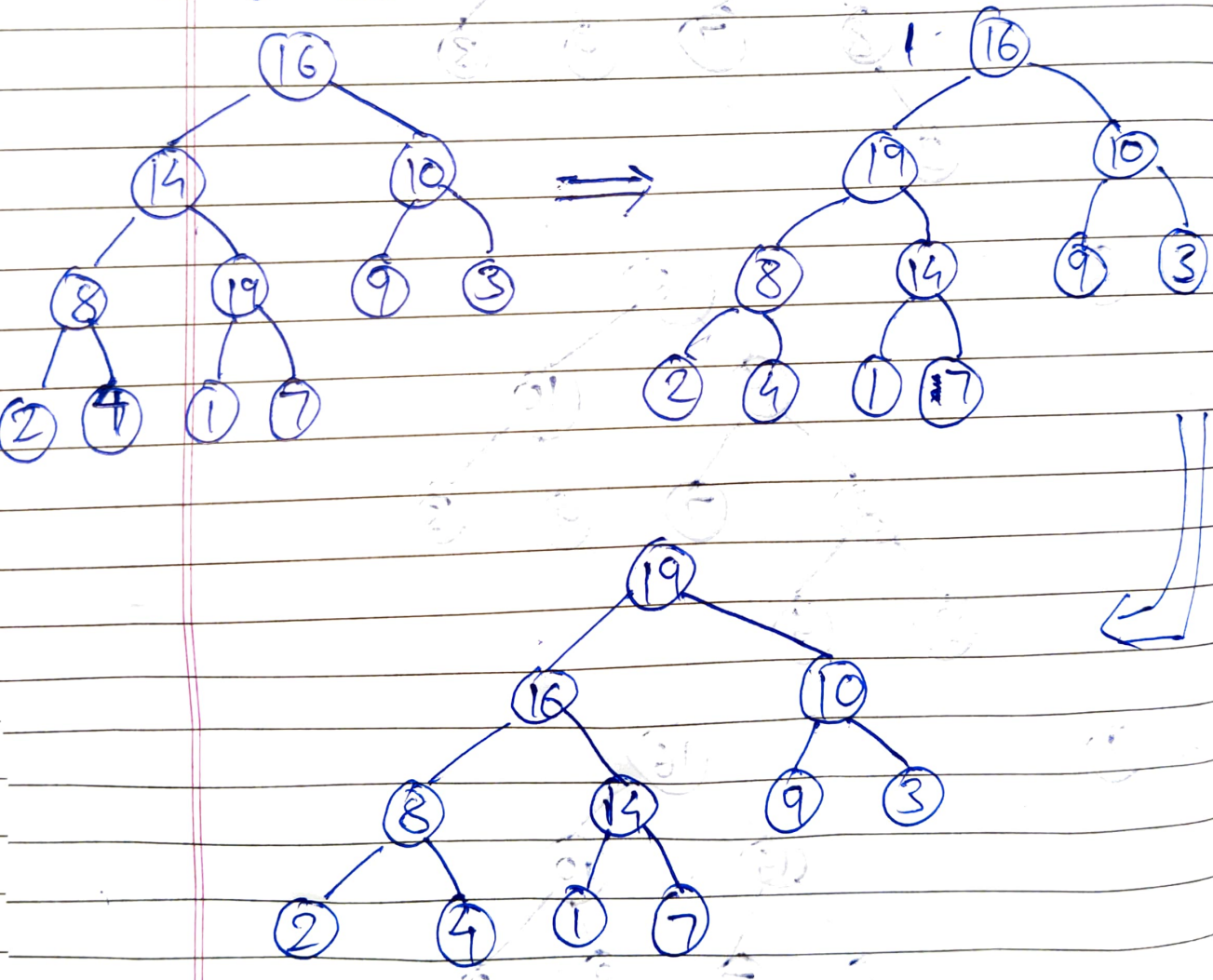
Add (1)



Add (19)



Heapifying:



New order of elements in array A after 19

= [19, 16, 10, 8, 14, 9, 3, 2, 4, 1, 7]