

Question 1

- The two arrays [20, 13, 17, 12, 7, 8, 14, 5, 6, 1, 3, 4] and [20, 17, 14, 6, 13, 7, 1, 5, 4, 12, 3, 8] both represent a max-heap of the set of numbers {1, 3, 4, 5, 6, 7, 8, 12, 13, 14, 17, 20}
 - False
- Let $T(n) = T(\alpha n) + T((1 - \alpha)n) + cn^3$, where α is a constant in the range $0 < \alpha < 1$, c is a positive constant, and $T(1)$ is $O(1)$. Then $T(n)$ is $O(n^3 \lg n)$.
 - True
- For a hash table of size m , quadratic probing improves over both linear probing and double hashing in that $O(m^2)$ probe sequences are used rather than $O(m)$.
 - False
- Given two sorted list of size n each, computing the k^{th} smallest element in the union of the two lists takes at least $\Omega(n)$ time.
 - False
- Radix sort requires an in-place auxiliary sort procedure in order to work properly.
 - False

Question 5

Satchel's is considering opening a series of restaurants along a highway. The n possible locations are along a straight line and the distances (in miles) of these location from the start of the highway are given in increasing order $m_1, m_2, m_3, \dots, m_n$. For each location i , p_i is the expected profit for opening a restaurant at location i . In addition we are given the following constraints -

- At each location, they can open at most one restaurant.
- Any two restaurant must be at least k miles apart, where k is a positive integer.

Design and analyze an efficient algorithm to compute the maximum expected total profit and the corresponding locations subject to the given constraints.

Solution

First, we pre-compute the nearest location j before location i which is at least k miles away, where $1 \leq i \leq n$ and use -1 if there is no such location. Let assume that `NearestNonConflict[i]` holds this value.

Let , `MaxProfit[i]` is defined as the maximum profit using first i locations. Now we have the following-

- If `NearestNonConflict[i] = -1`, then $\text{MaxProfit}[i] = \max(p_i, \text{MaxProfit}[i-1])$
- Otherwise, $\text{MaxProfit}[i] = \max(\text{MaxProfit}[\text{NearestNonConflict}[i]] + p_i, \text{MaxProfit}[i-1])$

For the first case, optimality holds, because if location i is included in optimal solution then optimal profit till location i is p_i , otherwise optimal profit is $\text{MaxProfit}[i-1]$. Similarly for the second case, if location i included in optimal solution then optimal profit is $(p_i + \text{MaxProfit}[\text{NearestNonConflict}[i]])$, otherwise optimal profit is $\text{MaxProfit}[i-1]$. In each case, cut and paste argument could be used to see the optimal substructure property.

Now, we can use following bottom-up strategy to compute the above recurrence,

```
for i = 1 to n
  if NearestNonConflict[i] == -1
    MaxProfit[i] = max (pi, MaxProfit[i-1])
  else
    MaxProfit[i] = max (MaxProfit[NearestNonConflict[i]] + pi, MaxProfit[i-1]);
```

Complexity analysis: Computing the nearest location takes $O(n)$ time and computing maximum profit takes $O(n)$ time. So, complexity of the algorithm is $O(n)$

Computing the Optimal Locations: Using Backtracking we can compute optimal locations.

We check Whether $\text{MaxProfit}[i] = \text{MaxProfit}[i-1]$.

If yes, then, we do not open restaurant at location i and consider the solution for the first $(i-1)$ locations.

Otherwise, we open restaurant at location i and consider the solution of first `NearestNonConflict[i]` locations.

Note: Any solution not in $O(n^2)$ time, is not considered as efficient solution and partial credit is given for those solutions.

Grading Policy:

- 8 points: Writing the recurrence (notation, equation)
- 4 points: Design Strategy (Top-Down or Bottom-Up) to solve the recurrence [may be code or may be some description]
- [2+2] points: Complexity Analysis and Optimal Substructure.
- 4 points: Constructing the solution / getting the optimal locations.