Report
Assignment 3

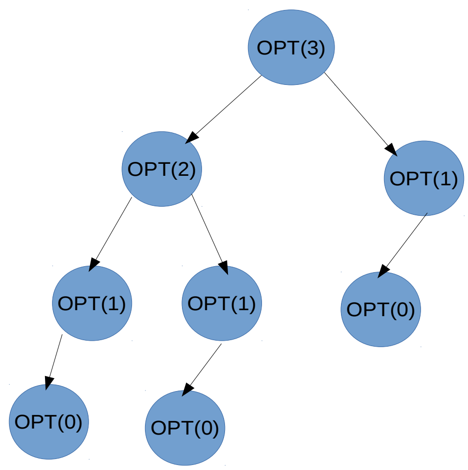
Name - Parul Bansal
Roll No - B14116

Question 1 (Weighted Interval Scheduling)

Psuedo Code of Compute-Opt(j)

```
If j = -1
  return 0
Else
  return max(vj + Compute-Opt(p(j)) , Compute-Opt(j-1))
Endif
```

Recursion Tree



Observations

- Time complexity of above algorithm is exponential.
- Time complexity is more because of the spectacular redundancy in the number of times it issues each of these calls.

Psuedo Code of M-Compute-Opt(j)

```
If j = -1
 return 0
Else if M[j] is not empty
  return M[j]
Else
  M[j] = max(vj + M-Compute-Opt(p(j)) , M-Compute-Opt(j-1))
   return M[j]
Endif
```

Observations

- Time complexity of above algorithm is O(n^2) .
- The running time of M-Compute-Opt(n) is O(n) (assuming the input intervals are sorted by their finish times).
- The running time for sorting the input intervals by their finish times is O(nlogn).
- The running time for calculating the function p(j) for each interval j is O(n^2).
- Memoization strategy is used in above algorithm and it is more efficient compared to the first one.

Psuedo Code of Find-Solution(j)

```
If j = -1
  Ouput nothing
Else
```

```
 If  v_j + M[p(j)] >= M[j-1] then
      Output j together with the result of Find-Solution(p(j))
  Else
      Output the result of Find-Solution( j − 1 )
 Endif
Endif
```

Observations

- Find-Solution returns an optimal solution in O(n) time (assuming the array M of the optimal values of the sub problems is given).
- Find-Solution *traces back* through the array M to find the set of intervals in an optimal solution.

Question 2 (Counting Inversions)

Psuedo Code of Merge-and-Count(A,B)

```
Maintain a Current pointer into each list, initialized to
  point to the front elements
Maintain a variable Count for the number of inversions,
  initialized to 0
While both lists are nonempty:
   Let a_i and b_j be the elements pointed to by the Current pointer
   Append the smaller of these two to the output list
   If b_j is the smaller element then
        Increment Count by the number of elements remaining in A
   Endif
  Advance the Current pointer in the list from which the
    smaller element was selected.
EndWhile
Once one list is empty, append the remainder of the other list
   to the output
Return Count and the merged list
```

Psuedo Code of Sort-and-Count(L)

```
If the list has one element then
   there are no inversions
Else
    Divide the list into two halves:
    A contains the first ceiling(n/2) elements
    B contains the remaining floor(n/2) elements
    (r_A , A) = Sort-and-Count(A)
    (r_B , B) = Sort-and-Count(B)
    (r , L) = Merge-and-Count(A, B)
Endif
Return r = r_A + r_B + r , and the sorted list L
```

Observations

- Time complexity of above algorithm is O(nlogn).
- It is almost same as merge sort.
- The Sort-and-Count algorithm correctly sorts the input list and counts the number of inversions; it runs in O(nlogn) time for a list with n elements.
- Divide and Conquer Strategy is applied in above algorithm.

Question 3 (Closest-Pair)

Psuedo Code of Closest-Pair(P)

Closest-Pair(P)

  Construct P_x and P_y (O(n log n) time)

  (p_0^* , p_1^*) = Closest-Pair-Rec(P_x , P_y)

Closest-Pair-Rec(P_x , P_y)

If $|P| \leq 3$ then

find closest pair by measuring all pairwise distances
Endif

Construct Q_x , Q_y , R_x , R_y (O(n) time)
(q_0^*, q_1^*) = Closest-Pair-Rec(Q_x , Q_y)
(r_0^*, r_1^*) = Closest-Pair-Rec(R_x , R_y)

$\delta = \min(d(q_0^*, q_1^*), d(r_0^*, r_1^*))$
x^* = maximum x-coordinate of a point in set Q
$L = \{(x, y) : x = x^*\}$
S = points in P within distance δ of L .

Construct S_y (O(n) time)
For each point $s \in S$ y , compute distance from s
to each of next 15 points in S y
Let s , s' be pair achieving minimum of these distances
(O(n) time)

If $d(s, s') < \delta$ then
Return (s , s')
Else if $d(q_0^*, q_1^*) < d(r_0^*, r_1^*)$ then
Return (q_0^*, q_1^*)
Else
Return (r_0^*, r_1^*)
Endif

Observations

- The running time of this algorithm is O(nlogn).
- The above algorithm divides all points in two sets and recursively calls for two sets.
- After dividing , it finds the strip in O(n) time.
- Also, it takes O(n) time to divide the P_y array around the mid vertical line.
- Finally finds the closest points in strip in O(n) time.
- Divide and Conquer Strategy is applied in above algorithm.

Question 4 (Segmented Least Squares)

Psuedo Code of Segmented-Least-Squares(n)

```
Array M[0 . . . n]
Set M[0] = 0
For all pairs i ≤ j
        Compute the least squares error e_{i,j} for the segment p_i , . . . , p_j
Endfor
For j = 1, 2, . . . , n
        M(j) = min((e_{i,j}+ C + M[i-1]) forall 1 ≤ i ≤ j)
Endfor
Return M[n]
```

Psuedo Code of Find-Segments(j)

```
If j <= 0 then
        Output nothing
Else
        Find an i that minimizes e_{i,j} + C + M[i − 1]
        Output the segment {p_i , . . . , p_j} and the result of
        Find-Segments(i − 1)
Endif
```

Observations

- The running time of this algorithm is O(n^2) once all the $e_{i,j}$ values have been determined..
- The total running time to compute all $e_{i,j}$ values is O(n^2).
- Dynamic Programming strategy is applied in above algorithm.