Homework 1

B10902033 林祐辰

References:
Problem 1: B10902032 李沛宸
Problem 2: B10902032 李沛宸
Problem 3: B10902032 李沛宸, Leetcode 1505
Problem 4: B10902032 李沛宸
Problem 5: B10902032 李沛宸, B10902037 羅煜翔
Problem 6: B10902032 李沛宸, B10902037 羅煜翔

Problem 5:
(a) Optimal choice: picking the construction M0, M1, M4, M6, with the cost of 4, 5, 2, 3, and the total cost 14.
(b) Algorithm:
First sort t array in an ascending order (bind with p). Next, insert W.p in a min_heap in order, and when W.t < T[++i] we add extractMin(min_heap) to min_cost.After i adding up to N, min_cost will be the answer.
Time complexity:
sort -> O(M log M), insert W.p in min_heap at most M times -> O(M log M), extract min from min_heap N times -> O(N logN).Total time complexity = O(M log M).
Correctness:
The substucture of this problem is N-1 with M-1 wich means that one weapon defend one wave. And by choosing greedily that is to choose the cheapist weapon before the deadline, we can maintain the optimal substructure.
(c) Optimal choice: picking the construction M5, M1, M2, M7, M3, with the cost of 3, 1, 2, 6, 5, and the total cost 17.
(d) Algorithm:
First sort t array in an ascending order (bind with p).Insert all p into a min heap, of which T[i] < t < T[i+1].And we use dp[T][K] to store the min cost for T attack, K weapon storage.To set the dp table we first compute each heap seperatly which for T=i dp[i][0] = 0, dp[i][k] = dp[i][k-1] + extract_min(min_heap), then we merge the seperate heap by dp[T][K] = min(dp[T-1][0~K], dp[T][K~0]).Lastly the answer will be dp[T][K].
Time complexity:
sort -> O(M log M), traverse + insert heap -> O(M log M), set dp table -> O(MK), extract in dp -> O(M log M). Total time complexity O(MK + M log M).
Correctness:
When we set up the first dp table store the cheapest way to store K weapons for T[i] < t < T[i+1], so after modifying the table we can get the min cost.
(e) pass
(f)Algorithm:
Similar to 5(b) but this time instead of inserting p we insert p-t, and when adding up min cost we need to add up T (the day wave arrive), so that (T-t) will the extra money to pay.
Time complexity:
Same with 5(b) since the operation we add is O(1).
Correctness:

Same with 5(b) just change from choosing the cheapest p to p+(T-t).

Problem 6:

(a) Algorithm:

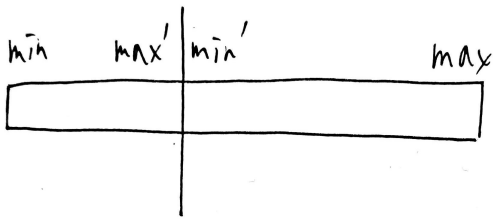
```
min_total_speed_dif(V[N]){
    sort(V)  //ascending order
    min_dif = V[N-1] - V[0]  //max - min
    int V_dif[N-1]  //difference for every adjancent element
    for(i = 0 to N-2){
        V_dif[i] = V[i+1] - V[i]
    }
    sort(V_dif)  //descending order
    int ans[N]
    for(i = 0 to N-2){
        ans[i] = min_dif
        min_dif -= V_dif[i]
    }
    ans[N-1] = min_dif

    return ans
}
```

The time complexity is O(n log n), since we do two sort and two for loop, which is O(2 * (n log n) + 2 * n) = O(n log n).

The algorithm works by choosing greedily, which is to cut the group at largest V_diff.

Brief proof:



problem : min_total_speed_dif = max - min

Sub_problem : min_total_speed_dif = max - min' + max' - min

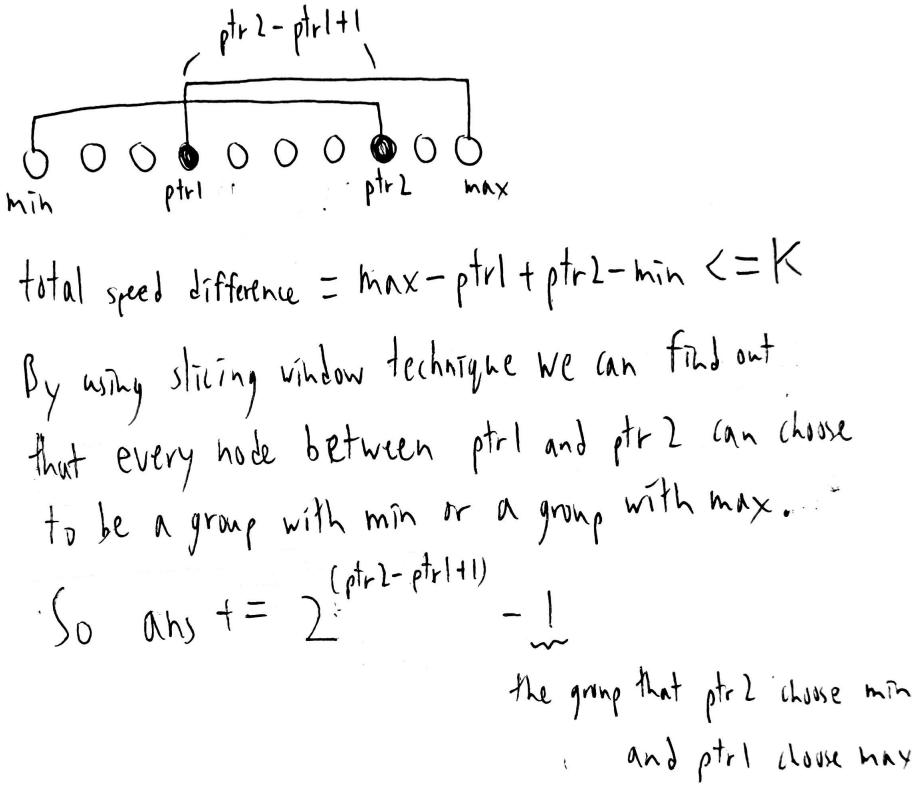
= max - min - (min' - max')

V_dif

(b) Algorithm:

```
ways_to_divide(V[N], K){
    sort(V)  //ascending order
    max_dif = V[N-1] - V[0]  //max - min
    int ans = 0
    for(i = 0 to N-2){
        if(max_dif - (V[i+1] - V[i]) <= k){
            ans++
        }
    }
    int ptr1, ptr2 = 0  //pointer for sliding window
    while(ptr1 < N-1){
        if(ptr2 < N-1 and max_dif + V[ptr2+1] - V[ptr1] <= k){
            ptr2++
        }
        else{
            ans += 2^(ptr2-ptr1+1) - 1  //calculate ans by using sliding window
            ptr1++
        }
    }
    return ans
}
```

Time complexity:
sort -> O(N log N), for loop -> O(N), while loop -> O(N).Total time complexity = O(N log N).
Brief explain of sliding window technique:



Correctness:
The algorithm works by first calculate the ones that are divided in ordered groups, than calculate the ones

that are divided in unordered groups and by simple calculate and sliding window technique we can be sure to seperate two groups with out missing or overlapping.

(c) 15 Groups:

Group by 4: {{1, 2, 3, 4}}

Group by 1: {{1}, {2}, {3}, {4}}

Group by 2, 2: {{1, 2}, {3, 4}}, {{1, 3}, {2, 4}}, {{1, 4}, {2, 3}}

Group by 1, 3: {{1}, {2, 3, 4}}, {{2}, {1, 3, 4}}, {{3}, {1, 2, 4}}, {{4}, {1, 2, 3}}

Group by 1, 1, 2: {{1}, {2}, {3, 4}}, {{1}, {3}, {2, 4}}, {{1}, {4}, {2, 3}}, {{2}, {3}, {1, 4}}, {{2}, {4}, {1, 3}}, {{3}, {4}, {1, 2}}

(d)

1.Subproblem: dp[i][j], which stores the ways for i people divide into j groups.

2.Recurrence relationship: dp[i][j] = dp[i-1][j-1] + j * dp[i-1][j]. For i in [0, N], j in [0, N], dp[i][0] = dp[0][j] = 0

3.Correctness: dp[i-1][j-1] means i people choose to be a group it self, j * dp[i-1][j] means i people choose to join in another group(j choices).Because every people will only have this two choices so the dp table will store all the answer for the ways to group N people into any number of groups.

Space complexity: O(N ^ 2) for dp[N][N].

Time complexity: O(N ^ 2) since every element in the table can be compute in O(1).

(e) 9 Groups:

Group by 1: {{1}, {2}, {3}, {5}}

Group by 2, 2: {{1, 2}, {3, 5}}

Group by 1, 3: {{1}, {2, 3, 5}}, {{5}, {1, 2, 3}}

Group by 1, 1, 2: {{1}, {2}, {3, 5}}, {{1}, {3}, {2, 5}}, {{1}, {5}, {2, 3}}, {{2}, {5}, {1, 3}}, {{3}, {5}, {1, 2}}

(f)

1.Subproblem: dp[i][j][k] which stores the ways for i people divide into j groups and there total speed difference <= k.