**YUELING QIN**

**Problem Set 4**

**Problem1.Placing base stations**

--We start from eastern home endpoint, and keeping track the distance to western endpoint. To do that, find a point located first house, which the distance equal to 4 mile, at that time, we can set a base station to this point. Then we need to remove all houses that occurred within 4 miles for this station.

The Algorithm is using greedy strategy, which we do best for this design.

--To proof the correctness:

For example, we got solution for this problem:

A: is represent the cell phone station, it’s a list {a1, a2, a3, a4…..an} by using greedy algorithms.

S: is a optimized solution, list {s1, s2, s3, s4, s5……sx)which have same size with A

Proof: Induction

Base one: distance(a1) >= distance(s1) based on what we choose to set,

We choose exactly 4 miles, which is most farthest distance we can set.

Case n: if we have distance(an)>= distance(sn) now, from my algorithm, I will get solution that all house from a1 to sn have already covered by s1 to sn stations.

Case n+1: based on last assumption, we can apply the rule to n+1 case, the next station we need to set is 4 miles, since it’s not covered yet, we need to add it. Then, we get final result for this addition, distance(an+1) must lager or equal w(sn+1).

Greedy algorithm: example A is best way solving this problem. we get the small number of station, because we always set the farther distance between houses.

The distance /an = number of station.

Worst-case analysis:

Because we just need to set he station, we don’t need to sort the distance, we just go through all house in the county form one endpoint to other one. And remove the house within 4 files. So, the total worst-case running time should be .

**Problem2.Planing an expedition**

Given edge (v,e) , we starting from point ‘s’ and time taken to travel from s

To v along the fastest path denotes d(v). Search the analogous to shortest-path

Distances, we have d(v) <= f(d(u)) for every edges(u,v).

We will maintain a set S of explored nodes, the ones for which have correctly calculated d(v), and maintain a label l(v) for the nodes not in S.

Now update l(v) based on the above rule but only consider edges (u,v)

Where u belong to S and set the node w not belong to S with smallest l () value and

Add it to the set S and repeat

S<---{s} with d(s)=0

set l(v) = infinite not belong to S

Let v\* = s (v\* denotes the last node added to S)

While s not equal V

For every edge e = (v\*, v) where v not belong to S

Update l(v) <-- min{l(v), f(d(v\*))}

Let w not belong to S be such that l(w) = min l(v)

Set s <--- SU{w} , d(w) = l(w), v\* = w.

For note that there is an s-> w path p, such that the time taken to travel along P is

d(w) and d(w) = l(w)= min f(d(u’)), So, if u’ belong to S is the node that attains the minimum in l(w), then p is obtained by adding the edge (u’, w) to the path P.

Now we show d(w) is indeed the least time taken to reach w along any s->w path.

So, consider any s->w path p. let v be the first node on P that is not in S, and u belong to S be the node on P just before v. Let p’ be the portion of path P form s to v. Let t1 and t2 be the times at which we reach u and v respectively.

Correctness: induction:

Base case: S =1 is trivial

Case n: assumption S= m where m bigger or equal 1

Case n+1: based on case n assumption, we have node v add to S,

Set the edge u-v chosen edge. The shortest path S-u add edge u-v is path from node s to v which is π(v). And we cannot find any path is shorter than π(v). Let v\*-s\* be the start edge in P that leaves S, and P’ is the subpath to s\*, because P already too long as soon as it leaves S.

We can the distance of P must bigger than the P’ add the distance between (v\*,s\*), also bigger than distance of v\* add the edge of v\*,s\*, so it must bigger than the π(v).

Worst-case Analysis:

Min: total,Delete node from a structure heap .Set the edge would be take time

The total running time will be

Where m>n

**Problem3**

Algorithm:

we have a list l1,l2,l3….. ,size is k, which is already sorted

Firstly, we merge l1 and l2, traverse l1, and merge l1, l2 with l3.

Each time the process take k-1, the running time we can compute by keeping this process at the end of list,

Function: a[(k-1)+(k-2)+…+1]=

Using the math rule, we can simplify to n(k\*(k-1))/2= we have now

So that is running time.

Divide-Conquer:

1. we divide to some parts: the list be divided to two parts,

then we have two new list, each one size is half of original one.

1. for each part, we do some thing at the same time, recursive merge for one new list, and other new list.
2. Finally, we need combined the two processed list together, we get final result of list.

Proof:

Worst-case:

Divide-and-Conquer algorithm, we divided whole part into two separate parts, and do recursive call in both side at same time, finally, we combined each result of recursive call. The worst case is same with

**Problem4**

Divide: There exist a slower slope line in the surface, on the other hand, we have faster slope line,

Recursion and Combined: use recursive call to find vidible line for all list.

we sort this line by using merger sort, l1, l2, l3,,,,, ln-1and m1, m2 ,m3, ..mv-1which is visible line. Then we look at the x-coordinates of intersection of each line pairs. X1, x2, x3…xn+v-2 It will take O(n) running time. Now, for each, we consider the line that is uppermost in L at x-coordinate xn, If the index of li that is closet to xn and to the left of xn is li-1, then the uppermost line in L at xn is L’. In this way, if index of mi that is closet to xn and to its right point in the plane at which l’ and l’’ interact. We established b\* lies between x-coordinates of xn-1and xn. After that, we need to get the next one of the recursive call.

Proof:

Basic conditions: only three rows, so that L1, L2, L3 is the line slope ascending order, then we know the L1 and L3 must be visible, L2 is visible, if and only if the intersection L1 and L3 in the right intersection. When we make the process of merging, we found that the point (b\*, c\*) of the intersection l and l’. We can know that li-1is the leftmost xn , and l of xn to the right, we delete all the rows, find new merged lines are visible lines.

Worest-case:

Divide: Constant time,

Recursion and Combined: running time O(n) for go through the all visible line, which size n. meger sort ,

)

**Problem5**

Fistly, start the local root in the binary tree, wen current is not null, we create a while loop function for it, then go to inside of while loop, comparing the left and right. If local value lager than left child of it, we set the current to leftchild of it.

Else local value lager than rightchild of it, we set to rightchild.

Since all values in the tree are distinct real value, there is a minimum number

among all the values. And the node with this value will be global minimum, which is also

A local minimum. local=root;

While(local!=Null) {

If ((local.value < local.leftchild.value) &&(local.value < local.rightchild.value))

Return local;

Else if (local.value > local.leftchild.value)

local=local.leftchild;

Else if (local.value > local.rightchild.value)

local=local.rightchild;

}

Worst-case analysis:

For a binary tree, the loop will take nlogn running time, and add some other constant running time, we can get