CMPT 307 ①

Quiz 3 Sol^{n}

1 (a) Since Min(·) & Max(·) routines take constant time to execute, the recurrence relation can be written as

$$T(n) = 2T(n/2) + O(1)$$
$$= 2T(n/2) + O(n^0).$$

Using Master theorem, we can write

$T(n) \in O(n)$. Here the # of subproblems ~~$(n^{\log_b a})$~~ $O(n^{\log_b a})$ dominates the running time.

(b) i) Using Master theorem we can write the closed form of $T(n) = 9T(n/3) + n^2$ as $T(n) \in O(n^2 \log n)$

ii) Consider the recursion tree:

The height of the tree is $(n-1)$

The amount of work done at level i is $O(i)$.

∴ Total cost

$$T(n) = \sum_{i=1}^{n-1} O(i)$$

which is $O(n^2)$.

∴ Algorithm (ii) is the most efficient.

2. Let $m = n/2$ be the middle index.

- If $A[m]$ is local minimum, report $A[m]$
- If $A[m-1] > A[m] > A[m+1]$, there exists a local minimum in the array $A[m..n]$
- If $A[m-1] < A[m] < A[m+1]$, there exists a local minimum in the array $A[1..m]$.

$\therefore$ using ~~atmost~~ $O(\log n)$ probes, one can find a local minimum in the array $A[1..n]$.

3. Closest pair problem

a) Let P be the set of points. Let P_x be the list of points of P sorted by x-coordinates. Let P_y be the list sorted by the y-coordinates. Let

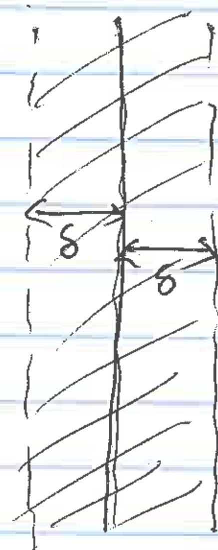
Q : first half of points of P_x
R : Second half of points of P_x.

With a single pass through the points P_x & P_y, we can build the following lists in $O(n)$ time:

Q_x : Points of Q sorted by increasing x-coordinate
Q_y : " " Q " " " y- "

R_x : " " R " " " x-coordinate
R_y : " " R " " " y- "

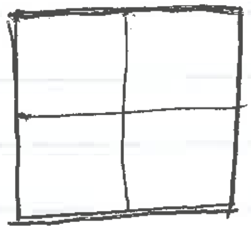
Recursively, we determine the closest pairs in Q + R. Let δ_1 & δ_2 be the closest pair distances of Q & R respectively.
Let $\delta = \min(\delta_1, \delta_2)$.
Let L be the dividing line of P_x into in Q_x & R_x.



L

Consider a strip of 2δ width around L. In the combine step, we need to find the closest pair less than δ with one point in each side of L. Thus we need to consider only the points in the shaded region.

Consider a window W of length $2\delta \times 2\delta$



There could be at most 12 points that could lie inside any ~~box~~ placement of W. (Why?)

If we sort the points in 2δ-strip,

we only check distances of those within 11 positions in the sorted list. The claim is (assuming s_i to be the i^{th} smallest y-coordinate point in 2δ-strip.)

Claim If $|i-j| \geq 12$, then the distance between s_i & s_j is at least δ.

$\therefore$ the recurrence relation of the above divide-and-conquer algorithm is

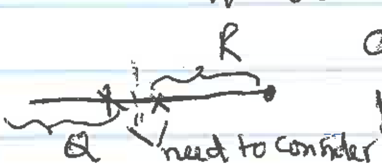
$$T(n) = 2T(n/2) + O(n)$$

whose closed form is $T(n) \in O(n \log n)$.

(b) When the points of P lie on a line, the window is an interval of length 2δ. Therefore, there is ~~only~~ at most one pair to consider. These points are ~~the two with~~

a) largest x-coordinate point of Q

b) smallest x- " point of R.



4. Observe that if $a_1 \neq 1$, it is not possible to make change to all possible amount. Therefore, we assume that $a_1 = 1$.

~~Then~~ Let $Opt(j)$ denote the smallest # of coins used that add up to j. The recursive definition of $Opt(j)$ is

$$Opt(j) = \infty \quad \text{if } j < 1$$

$$= \min_{1 \leq i \leq n} \{ 1 + Opt(j - a_i) \}$$

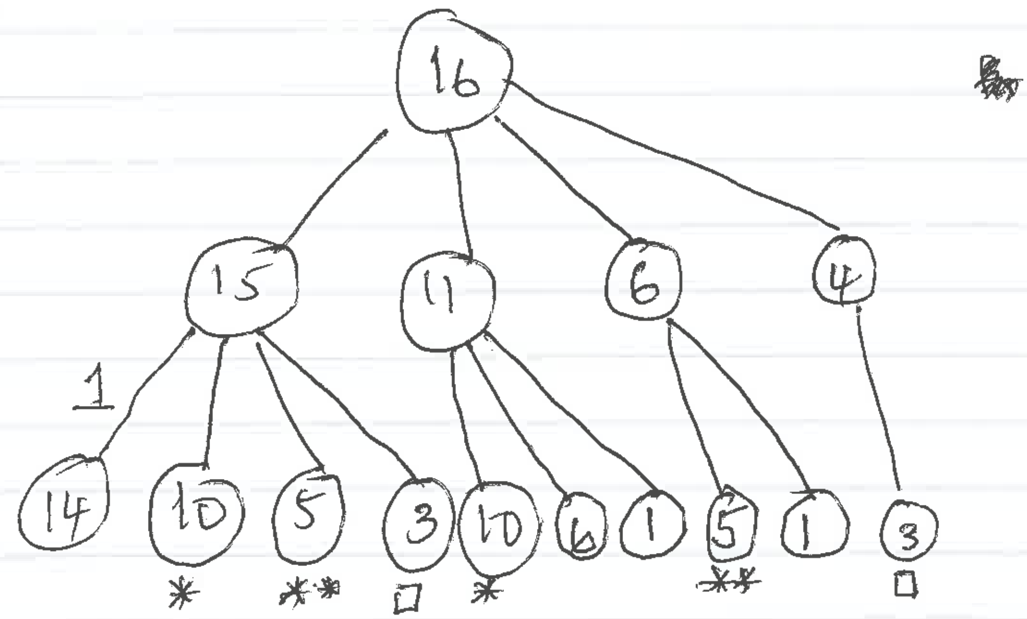
$Opt(C)$ will return the optimal result.

```
M-Coin-Change (j)
{
  if j < 0 then return ∞
    else
    {
      if M[j] ≠ null, return M[j]
      else
          M[j] = min { 1 + M-Coin-Change (j - a_i) }
                1≤i≤j
    }
}
```

We call M-Coin-Change(C).

The array M[.] has C entries.
Each entry requires $O(n)$ time to compute.
There are $O(C)$ subproblems in total.
Total cost is $O(nC)$.

(b)



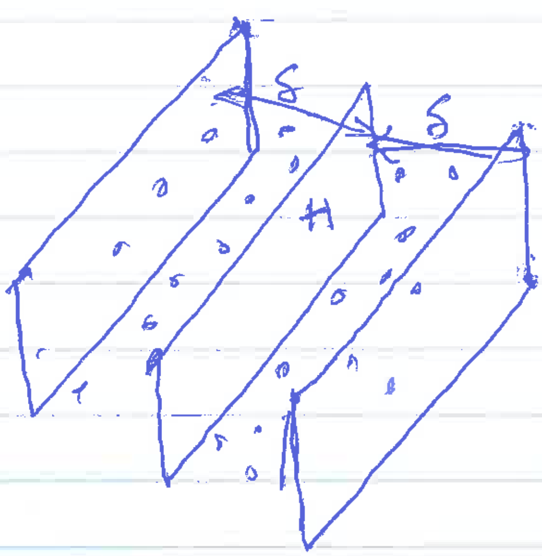
There are many instances of the same subproblem. The memoized version solves only once ~~instances~~ of such repeated problems instances recursively.

(5) The problem is discussed in the text, page 212.

(6) Closest-pair problem:

We apply the similar divide-and-conquer approach discussed for problem 3.
For 3-d case, the combine step is different.

- We divide P into Q & R by median plane H.
- Recursively, solve the problem in two halves. Let δ be the closest pair found so far.
- Consider all points P' within δ thick slab around H.



- If we place a cube of size $2\delta \times 2\delta \times 2\delta$ in the slab, we can show that at most 64 (O(1)) points of P' lie inside the cube.
- Therefore there are O(n) pairs of points of P' need to be considered.

- These $O(n)$ pairs can be determined in $O(n\log n)$. Using the sparsity property, this process can be solved in $O(n)$ time.

- Thus the divide-and-conquer approach can be solved in $O(n\log n)$ time.

(7) First note that if $\sum_i a_i$ is an odd number, the answer to the partition problem is "No".

Therefore, $M = \frac{1}{2}\sum_i a_i$ is an integer

We formulate the 2-partition problem as a knapsack problem as follows.

INPUT

Knapsack size $= M$

There are n items $\{1, 2, --, n\}$

item i has weight a_i

Output

A subset S of items so that $\sum_{i\in S} a_i \le M$

and subject to the condition that

$\sum_{i\in S} a_i$ is as large as possible.

Note that the answer to the partition problem is "Yes" if and only if $\sum_{i \in S} a_i = M$.

Let $$\mathrm{Opt}(i,j) = \begin{cases} \text{"Yes"} & \text{if } \exists\, S' \subseteq \{1,2,\dots,i\} \text{ s.t. } \sum_{k \in S'} a_k = j \\ \text{"No"} & \text{Otherwise.} \end{cases}$$

The answer to the partition problem is "Yes" if and only if $\mathrm{Opt}(n,M) =$ "Yes".

The recurrence relation can now be ~~written~~ written as

$$\mathrm{Opt}(i,0) = \text{"Yes"} \quad \text{for all } i \in \{1,\dots,n\}$$

$$\mathrm{Opt}(i,j) = \text{"No"} \quad \text{for all } i \text{ \& negative-valued } j.$$

$$\mathrm{Opt}(i,j) = \begin{cases} \text{"Yes"} & \text{if } \mathrm{Opt}(i-1,j) \text{ is "Yes" or } \mathrm{Opt}(i-1,j-a_i) \text{ is "Yes"} \\ \text{"No"} & \text{otherwise} \end{cases}$$

We used the following alternatives:

- a_i is in the solution.
- a_i is not in the solution

The running time of the memoized-version or bottom-up ~~vers~~ version is $O(nM)$.

(8) Opt(i) : shortest length path from v_1 to v_i

Recursive formula

$$Opt(i) = 0 \quad \text{if } v_i \text{ has no incoming edge}$$

$$= \min_{\substack{\text{over all incoming} \\ \text{edges } (k,i)}} \{ Opt(k) + 1 \}$$

The memoized version of the above formula can easily be written.

The bottom-up version is also easy

```
A[1] = 0
for i = 2 to n do
{
   if v_i has no incoming edge, skip the following
   A[i] = ∞
   for ∀ incoming edge (k,i) to i
      if (A[i] > A[k]+1) then A[i] = A[k]+1
}
```

Since all incoming edges to the vertices are examined once, the running time is $O(|E|+|V|)$.