2019 ADA miniHW 2

b07902064 資工二 蔡銘軒

October 2, 2019

(a)

```
Closest-pair(P):
  if |P| <= 3:
    Enumerate all pairs and return the pair with shortest distance
  else:
    Find a vertical line L such that the number of points on the left
        plane is equal to that on the right plane
    left-pair = Closest-pair(left plane)
    left-min = Distance(left-pair)
    right-pair = Closest-pair(right plane)
    right-min = Distance(right-pair)
    delta = min(left-min, right-min)
    Remove points that are delta or more away from L, and denote the
        remaning points as p1, p2, ..., pn according to y coordinate
    For i in 1 to n:
      For j in i + 1 to i + 7:
        if Distance(pi, pj) < delta:
          Update delta and the closest pair
    return the closest pair


Closest-pair-on-cylinder(C):
  Split the surface into a h by (2 * pi * r) plane by a vertical line
      on the surface that does not cross any point
  Label the points with appropriate x and y coordinates
  Sort points by x and y coordinate and store in Px and Py
  plane-pair = Closest-pair(the generated plane)
  delta = Distance(plane-pair)
  Append the points with x coordinate in [0, delta] to the right end of
      the plane by incrementing their x coordinates by (2 * pi * r)
  Remove points that are delta or more away from x = (2 * pi * r), and
      denote the remaining points as p1, p2, ..., pn according to y
      coordinate
  For i in 1 to n:
    For j in i + 1 to i + 7:
      if Distance(pi, pj) < delta:
        Update delta and the closest pair
```

```
return the closest pair
```

(b)

The difference between a plane and a cylinder is that if we represent the surface of the cylinder as a plane, the right end and the left end is actually connected. This issue can be addressed by concatenating two planes so that it "wraps around".

We first run the "closest pair" algorithm on the surface of the cylinder as if it is a normal plane. The correctness of this algorithm is proven in the lecture, so we'll take it for granted. Let δ be the distance of the closest pair returned by the algorithm. Then we consider the "wrapping around" issue. Clearly, we only have to concatenate the plane with the part that satisfies $x \in [0, \delta]$ since the answer is certainly less than or equal to δ. When dealing with the newly concatenated part, we use the "merge" method in the "closest pair" algorithm and consider the points with $x \in [2\pi r - \delta, 2\pi r + \delta]$.

The "wrapping around" issue is then solved since we have considered the situation where the left end is actually connected to the right end. So we can conclude that the answer has considered all possible situations and thus is correct.

Side note: When transforming the surface to a plane, we choose a line that does not cross any point so we don't have to deal with the situation where a some points are on both ends.

(c)

We first analyze each step of "Closest-pair-on-cylinder".

1. Tranforming the surface to a plane and label them with x and y coordinate takes $O(n)$ steps.
2. Sorting the points by x and y coordinate takes $O(n \lg n)$ steps.
3. Calling the "Closest-pair" function is analyzed below.
4. Concatenating the planes and label the points with new appropriate coordinate takes $O(n)$ steps.
5. Using the "merge" method to calculate the answer in the concatenated part takes $O(n)$ steps.

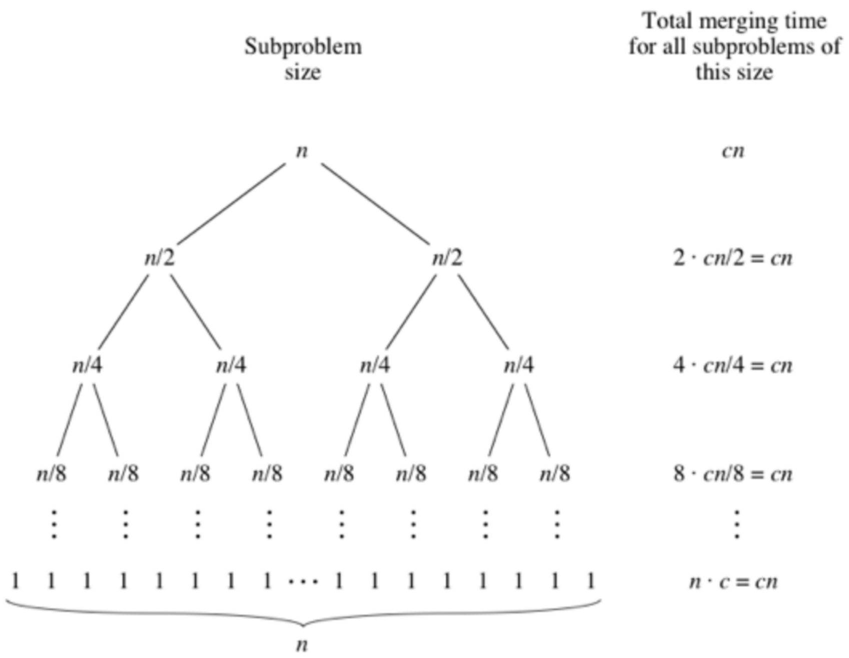
Let the running time of "Closest-pair" be $T(n)$, and its analysis is as follows:
Base case: $O(1)$
Recursive case:

1. Finding the line that splits the plane in half takes $O(n)$ steps.
2. Solving for the left and right planes takes $2T\left(\frac{n}{2}\right)$ steps.
3. Merging the result takes $O(n)$ steps.

With the above analysis, we can write down the formula:

$$T(n) = \begin{cases} O(1) & if\ n \le 3 \\ 2T\left(\frac{n}{2}\right) + O(n) & else \end{cases}$$

and use recursion-tree in the figure below to find out the time complexity.



Since the problem size is halved every level down the tree, it takes $\lg n + 1$ levels (starting at 0) to reach the base case.

We then consider the running time at every level. At each level, we have $2^i \cdot \frac{cn}{2^i} = cn$, where i is the level down the tree, and c is some constant > 0 for the $O(n)$ in the recursive case.

For the total $\lg n + 1$ levels, we have $cn \cdot (\lg n + 1) = O(n \lg n)$, which is the time complexity for the "Closest-pair" function.

Overall, we have $O(n) + O(n \lg n) + O(n \lg n) + O(n) + O(n)$, which is eventually $O(n \lg n)$

reference

Discussion with b07902075 林楷恩 and b07902132 陳威翰 and b07902028 林鶴哲
Pseudo code inspired by lecture slide
Picture taken from https://www.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/analysis-of-merge-sort