Algorithms TIN092

Yet another exercise session

Divide & Conquer recap

Similar to Dynamic Programming, we consider sub-problems, yet problems are smaller and usually do not intersect

Q: Oh, no, recurrences again?
A: Yes, but they pop up in a different place. We will need them in algorithm's running time evaluation.

Q: But we don't need to solve them exactly, right?
A: No, we do, since we need to have a good upper bound class for running time.

Q: What do I do, I don't know how to solve recurrences.
A: There will be a brief recap of how-to. There is also **Master theorem**.

How-to D&C

Algorithm:

1. Divide: split the problems into several sub-problems
(usually, a dull preparatory part)
2. Conquer: recursively solve the sub-problems

3. Combine: use solutions of sub-problems to solve the original one.
(usually, the most interesting part)

Analysis:

1. Correctness proof (“Combine” step mainly,
argument is understandably quite informal),

2. Running time evaluation (Recurrences again).

Example 1: Binary Search

Input:
Sorted array a[] of n integers,
Integer value v,

Output:
any i, if a[i]=v,
NO, if such i does not exist

Algorithm:
Divide: if a[1]>v, a[n]<v answer is NO;
split array into two arrays a[1:n/2-1], a[1:n/2+1];
Conquer: check whether the value is in one of the halves;
Combine: if both answers are NO, then NO;
otherwise, give index I from one of two halves;

Example 1: Binary Search

SDP(Sufficiently Detailed Pseudo-code):

```
BinarySearch(array a[], integers v, start) {
    if (a[1]>v or a[n]<v)                    (Divide)
        return NO;
    if (n==1) then
        return start;                    (current index in the larger array)
    else
        size=end-start;
        i=BinarySearch(a[1:n/2],v,start);                    (Conquer)
        j=BinarySearch(a[n/2+1:n],v,start+n/2);
        If (i or j is not NO) then return it, else return NO; (Combine)
     fi
}
```

Note that we skipped the obvious details, i. e. how we exactly deal with indices or the last if condition, as they take too much unnecessary space.

Example 1: Binary Search

Analysis

1. Correctness:

Finiteness is obvious (array size reduces with each recursive call, on n=1 termination, thus execution tree will have finite depth)

Output correctness:
Inductive: **Base**: n=1 – output is correct,
Step: n<k output is correct, for n=k+1:
From induction assumption, *Conquer* step returns correct answers for sub-problems. The problem admits a solution if and only if at least one of the sub-problems admits a solution.
Thus, if “NO” is returned, it is returned correctly.
If “i” is returned, it was an output of one of the sub-problems, thus, by assumption, a[i]=v, and it is also returned correctly.

Example 1: Binary Search

Analysis:

2. Running time evaluation:

T(n) <= 2 * T(n/2) + C

Consider the tree of recursive calls, it's depth is <= log(n)+1,
(why log(n)? try to draw the tree for any n=2^a)

Upon visiting each node, a constant time is wasted.

What is a total number of leaves in such a tree?.. *(n)*

T(n) takes no more than #leaves*C time, i.e. is of order ?.. *O(n)!*

Example 1: Binary Search

Better analysis:

Better bound can be established:
T(n) <= T(n/2) + C, as running time of both halves is not quite the same (one will terminate abruptly).

The tree of recursive calls is reduced to a path.

What is a total number of leaves in such a tree?..
(equal to depth, which is O(log(n))

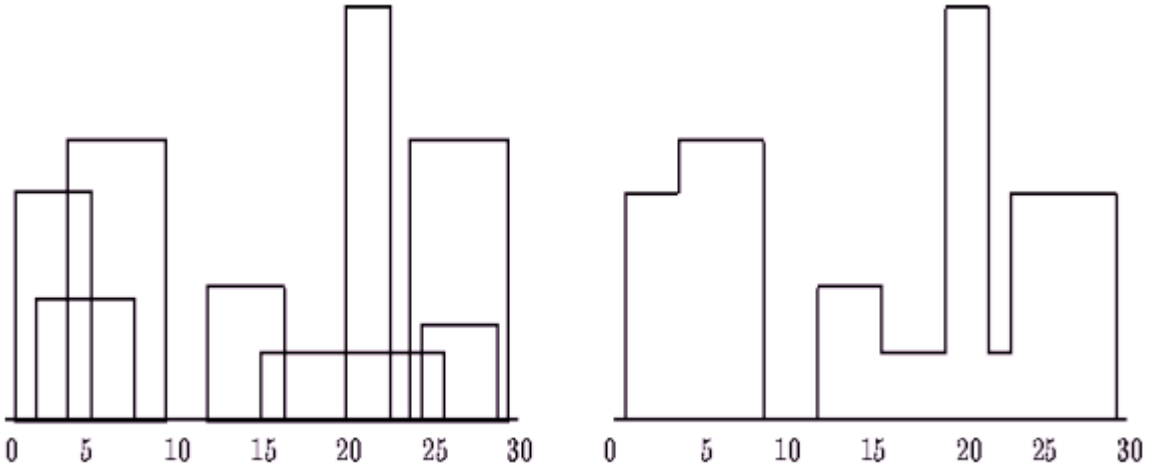
Same method:
T(n) takes no more than #leaves*C time, i.e. is of order *O(log(n))*.

(was O(n) before, pay attention, that O(n) bound is also **correct**, it's just not good enough)

Example 2: Skyline

Input:
Array L[] of n integers, with starting points of buildings
Array R[] of n integers, with ending points of buildings
Array H[] of n integers, with heights of buildings

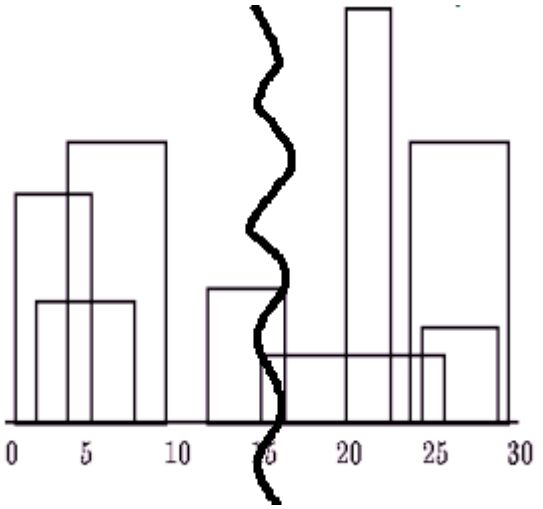
Output:
Array C[] of 2n integers, with coordinates.
Array D[] of 2n integers, with corresponding heights.



Example 2: Skyline

2n coordinates is sufficient, why – exercise.

Idea: spoil the picture with an ugly line,



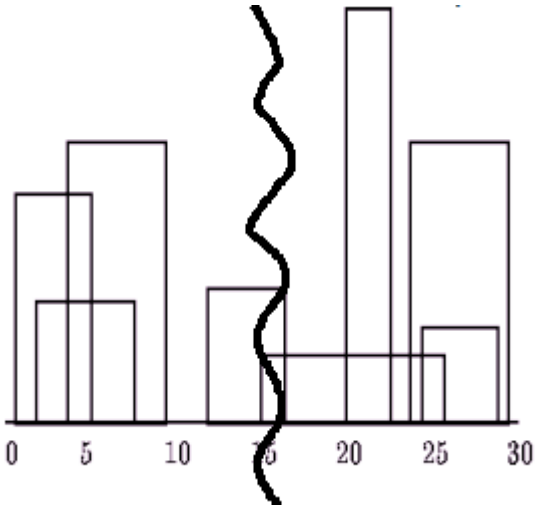
(Divide&Conquer)
Solve the sub-problems for
1)buildings to the left,
2)buildings to the right.

(Combine)
Merge skylines, decide the height of middle point.

Potential problems:
1) how to separate buildings to “left” and “right” buildings
2) how to decide the height of the middle point?

Example 2: Skyline

(Combine)
2 skylines (lists of coordinates).
How to merge?
Follow the lines by their x coordinate,
Choose the highest height among 2
(MergeSort style)

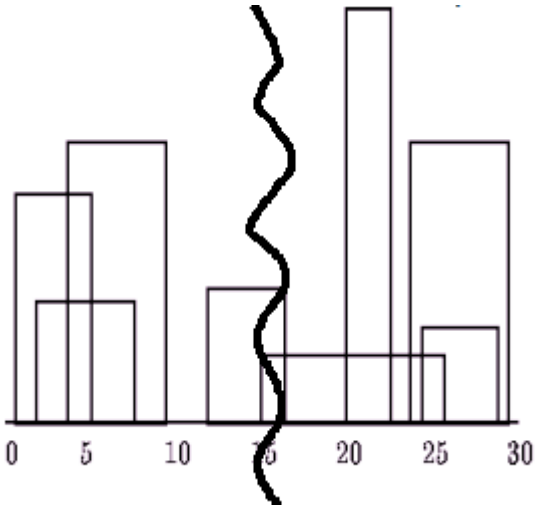


Potential problems:
1) how to separate buildings in “left” and “right” buildings

Example 2: Skyline

(Divide)

A set of buildings, how to separate it into 2 sets of buildings?



Sort buildings by left-most coordinate, (only once!)

Divide the resulting list into two, if possible.

If not, it's a single building, trivial task (n=1 – consider special case).

Good thing to think about:
when will we need to have the buildings sorted?

Example 2: Skyline

SDP:

MergeSort Array L[], rearrange indices of R[] and H[] correspondingly.
Return Skyline(L[],R[],H[]);

```
Skyline(Arrays L[],R[],H[] of n integers)
{
      if (n==1) then
          return {L[1],R[1]},{H[1],0};
      elseif
      return Combine(Skyline(L[1:n/2],R[1:n/2],H[1:n/2]),
                              Skyline(L[n/2+1:n],R[n/2+1:n],H[n/2+1:n]));
      endif
}
```

Continuation:

```
Combine(Arrays C1[], C2[], D1[], D2[] of n integers) {
    compare_append(i,j) {
        if (D1[i]>D2[j]) then
                append C1.[i] to C, D1[i] to D;
        elseif
                append C2.[j] to C, D2[j] to D;
        fi
    }
    i=j=1;D1[0]=D2[0]=0;
    while (i<=n and j<=n) do
        if  (C1[i]<C2[j]) then
            compare_append(i,j-1);
            i=i+1;
        fi
        compare_append(i-1,j);
        j=j+1;
    od
    Complement C and D with the rest of either C1,D1 or C2,D2;
    return C[],D[] without duplicate entries;
}
```

Example 2: Skyline

Analysis:

1. Correctness

Inductive proof,

Base n=1 – easy to check,

Step: assume output is correct for all n<k, for n=k:

Conquer step gives correct output by induction assumption,

Assume that real skyline has a different height somewhere.

According to induction assumption, either one skyline or another gives a correct height point. But we selected the highest of two → contradiction.

2. Running time evaluation

$T(n) = \text{Divide}(n) + \text{Conquer}(n) + \text{Combine}(n) = O(1) + 2T(n/2) + O(n)$,

(Combine(n) is of class O(n) because there are no more than 2n iterations of while (either i or j is decreased), each of which takes constant time.)