CS 5150/6150: Assignment 1
Due: Sep 23, 2010

Wei Liu

September 24, 2010

Q1: (1) Using master theorem: $a = 7, b = 4, f(n) = O(n)$. Because $f(n) = n^{\log_b a - \varepsilon}$ holds when $\varepsilon = \log_b a = \log_4 7$, we can apply the first case of master theorem and get $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{1.4})$.

(2)$a = 2$, $b = 2$, $f(n) = O(n \log^2 n) = n \log^2 n$. If we check the condition $f(n) = O(\log_b a \log^k n)$, we find it holds when $k = 2$. So $T(n) = \Theta(n^{\log_b a} \log^{k+1} n = \Theta(n \log^3 n)$.

(3) We can make a guess that $T(n) = O(n \log n)$, hence

$$\begin{aligned} T(n/\log n) &= \frac{n}{\log n} \log \frac{n}{\log n} \\ &= \log n \cdot \frac{n}{\log n} \cdot \log \frac{n}{\log n} + n \\ &= n \log \frac{n}{\log n} + n \\ &= n \log n - n \log \log n + n \\ &= O(n \log n) \end{aligned}$$

(4). $T(n) = \frac{\pi}{\sqrt{2}} \frac{T(1)}{T(2)} = \frac{\pi}{\sqrt{2}a} = \Theta(c)$.

Q2: We can do a merge sort using devide and conquer approach, which is $O(n \log n)$-time, and get a sorted array y first. Then to find two number y_i and y_j satisfying $y_i + y_j = x$, we do searching from both sides of the array y, that is, searching begin with $i = 1$ and $j = n$, here n is the array size. During the searching, if $y_i + y_j < x$, we update $i = i + 1$, else if $y_i + y_j > x$, we update $j = j - 1$, the algorithm will stop when $y_i + y_j = x$. This searching algorithm yields a $O(n)$ time. So the total computation complexity of the procedure is $O(n \log n) + O(n)$. Since $O(n \log n)$ is dominated, the algorithm

is $O(n \log n)$-time.

Q3: We need to look at the points where two lines meet, because these points seperate visible segment and invisible segments. The algorithm takes a few steps:

- Find all possible points where two lines meet. There is at most $\binom{n}{2}$ number of points, so this step takes $O(n^2)$.
- For each line equation, plug in all points in previous step, to see which side the points are, so we know if each points are uppermost. This is $O(n^2)$ points cross with n lines, so this step takes $O(n^3)$.
- Find the lines that meet at these uppermost points. These lines are what we need.

Q4: (a).One disadvantage of using a even number is there is no median for each subgroup, so we may need to compute the mean and use the mean instead. Another disadvange lies in that if using even number, the size of the subproblem will be bigger than that of using odd number. For example,for even number $2k$, the subproblem size is $T(\frac{4k-k}{4k}n) = T(\frac{3}{4}n)$, while for odd number $2k+1$, the subproblem size is $T(\frac{4k+2-(k+1)}{4k+2}n) = T(\frac{3k+1}{4k+2}n) = T((\frac{k}{4k+2} + \frac{1}{2})n)$. It is easy to see that $T((\frac{k}{4k+2} + \frac{1}{2})n)$ is smaller than $T(\frac{3}{4}n)$.

(b). Using 5 yields:

$$T(n)_5 = O(n) + T(n/5) + T(7n/10)$$

$O(n)$ is the time to compute medians, $T(n/5)$ is the time to compute pivot, and $T(7n/10)$ is the time for subproblem.

While using 3, we have:

$$T(n)_3 = O(n) + T(n/3) + T(2n/3)$$

$O(n)$ time to compute medians, $T(n/3)$ time to compute pivot, $T(2n/3)$ is the time for subproblem.

Since $T(n)_3 > T(n)_5$, we use 5 instead of 3.

(c). For using an odd number, eg, $m = 2k+1$, as we discussed before, the size of subproblem is $T((\frac{k}{4k+2} + \frac{1}{2})n)$. If increase the number from 5 to 7, 9, 11 or bigger, we can see that the size of subproblem increases, when $m = 5$, size of subproblem is $T(\frac{7}{10}n) = T(0.7n)$, when $m = 7$, size of subproblem is

$T(\frac{10}{14}n) \approx T(0.714n)$, when $m = 9$, size of subproblem is $T(\frac{13}{18}n) \approx T(0.722n)$. So using larger odd can't give us better result. On the other hand, as the increasing of the odd number, the time of computing medians will also increase. let's take 5 and 7 as examples: the time of computing medians are $\frac{n}{5}5\log 5 = (\log 5)n$ and $\frac{n}{7}7\log 7 = (\log 7)n$, we observe a increase of time when switch from 5 to 7. So in a sense, 5 if optimal.

Q5: Here is an example that the algorithm gives wrong answer.

	week 1	week 2	week 3	week 4
l	6	3	10	5
h	40	8	5	8

The algorithm will pick $l_1 = 6$ at week 1, because $l_1 + l_2 > h_2$.And actually choosing $h_1 = 40$ is a better solution in this case. So the problem of the algorithm is that it does not take h_1 into account, so even if in the following steps it chooses the optimal values, it can't get a global maximum in the end.

(b). The algorithm below modified the first step of the original one, which make sure that h_1 is taken into account:

StartIndex = 1;

If $h_2 \leq l_1 + l_2$ then

if $h_1 > l_1$ then

Output 'Choose a high-stress job h_1 in week 1'

Else

Output 'Choose a low-stress job l_1 in week 1'

Set StartIndex = StartIndex + 1;

Else

If $h_1 > h_2$ then

Output 'Choose a high-stress job h_1 in week 1'
Set StartIndex = StartIndex +1;

Else

Output 'Choose no job in week 1'
Output 'Choose a high-stress job h_2 in week 2'
Set StartIndex = StartIndex+2;

Endif

Do the original algorithm begin with i = StartIndex.

Q6:

```
Segmented-Least_Squares(n)
  Array M[0...n]
  Set M[0] = 0
  For all pairs i<=j
   Compute the least squares errors e_i,e_j for the segment p_i,...,p_j
  Endfor
  For j=1,2,...,n
  Use the recurrence (6.7) to compute M[j]
  Endfor
  Return M[n]
```

```
Find-Segments(j)
  If j=0 then
    Output nothing
  Else
    Find an i that minimizeds e_ij + C + M[i-1]
    Output the segment {p_i,....,p_j} and the result of
           Find-Segments(i-1)
Endif
```

This homework is a joint effort with Bo wang.