CS 4102 Written HW5 - Miscellaneous

Eric Xie

October 29, 2016

1.
The probability of the pivot being in the middle 50th percentile is 50%, since there is a 25% chance that the pivot is below the 25th percentile and 100-75=25% chance that the pviot is above the 75th percentile.
Since the base case does not select a pivot, we only need to find the probablity of this happening on the non-leaf nodes of the recursion tree.
Probability = 0.5^n , where n = # of internal nodes

If this were guaranteed, the worst case is that the pivot falls at the edge of the middle 50%, dividing the array into n/4 and 3n/4 values.
Since it takes linear time to perform pivots on each portion, the recurrence can be expressed as $T(n) = T(n/4) + T(3n/4) + n$
Using the substitution method, we assume that the runtime is $O(nlogn)$
That means there exists c and n_0 such that $T(n_0) \leq cnlog(n_0)$ This is true for $n_0 = 2$, $T(2) \leq c * 2 * 1$, when $c \geq 2$

For the inductive step:
$T(k) \leq cklog(k)$
$= T(k/4) + T(3k/4) + k \leq cklogk$
$= cklog(k/4)/4 + 3cklog(3k/4))/4 + k \leq cklogk$
$= c(\frac{k}{4}log(\frac{k}{4}) + \frac{3k}{4}log(\frac{3k}{4})) + k \leq cklogk$
$= \frac{ck}{4}(log(\frac{k}{4}) + 3log(\frac{3k}{4})) + k \leq cklogk$
$= \frac{c}{4}(log(\frac{k}{4}) + 3log(\frac{3k}{4})) + 1 \leq clogk$
$= \frac{c}{4}(log(k) - log(4) + 3log(3k) - 3log(4) + 1 \leq clogk$
$= \frac{c}{4}(log(k) - 2 + 3log(3) + 3log(k) - 6) + 1 \leq clogk$
$= \frac{c}{4}(4log(k) + 3log(3) - 8) + 1 \leq clogk$
$= clog(k) + \frac{3clog(3)}{4} - 2c + 1 \leq clogk$
$= \frac{3clog(3)}{4} - 2c \leq -1$
$= 3clog(3) - 8c \leq -4$

$c < 0.57$
No contradiction, so $T(n) \in O(nlogn)$

2.
At the last minute of any sequence, we can always fire the gun without consequence, since there is no point in charging the gun any further.
This will kill up to C(j) zombies, making the total kill count arr[currentTime-j].
Since we want to maximize total kills, we need to choose j so that C(j)+arr[currentTime-j] is maximized, checking every prior time.
Using the above properties, we can solve the problem using the following algorithm:

```
n=len[Z]
Initialize an array of size n to store the total zombies killed so far
for i in (0,n):
    if(i==0):
        return max(C(0), Z[0]);
    else:
       temp=arr[0];
       for j in (1,i):
          temp = max(temp, C[j]+arr[i-k]
       arr[i]=temp;
```

If we fire the gun, we can kill C[k] of the current zombies, with k being the last time the gun was fired. If we don't fire the gun, we are assuming that the **3.**
First, we must sort the list of lines, L, by increasing slope.
When the lines are sorted by slope, the first and last lines in the list are always visible.
In addition, when looking at a line, L_{n-2}, the last line, L_n, and any line in between, L_{n-1}
Let I_i be the point at which L_{n-2} and L_{n-1} intersect
Let I_j be the point at which L_{n-2} and L_n intersect
If the I_i is left of I_J, then L_n cannot fully dominate L_{n-1}, and all three lines are visible.
If the I_i is right of I_J, then L_n completely dominates L_{n-1}, and only L_{n-2} and L_n are visible.

Using the properties above, we can simply add subsequent lines to our list of visible lines, removing L_m as necessary to give the list of visible lines up to L_n.

```
visible = []
if(len(L)==1)
    visible.append(L[0])
else
    visible.append(L[0])
    visible.append(L[1])
for(int i=2;i<len(L);i++):
    n=len(visible);
    while(intersect(visible[n-2],visible[n-1]).x  < intersect(visible[n
&& n>1):
        visible.removeLast()
        n = n-1
    visible.append(L[i])
```

The runtime of this algorithm is O(n). At worst, the while loop has to remove every I_{n-1}, but once a line is removed it is never considered again, so the worst number of removals is n-2. The number of insertions is always n, so together the runtime is 2n-2, or O(n)