CSOR4231 Algorithm
HW2

Jialin Zhao — jz2862

February 19, 2018

1 Problem I

1.1 Show that when $p = 1/n$, the probability our sample is good is larger than some positive constant (independent of n).

According to the description of the question, the probability of X_1 = [R contains at least 1 element of T] as P_1. is exactly the same as 1 minus the probablity of the opposite event [R contains 0 element of S]:

$$P_1 = 1 - (\frac{n-1}{n})^n$$

The probability of X_2 = [R contains no element of S] as P_2 is :

$$P_2 = (\frac{n-1}{n})^n$$

Because X_1 and X_2 is independent events, the probability of the sample is good should be :

$$P = P_1 * P_2 = (1 - (\frac{n-1}{n})^n) * (\frac{n-1}{n})^n = (1 - a_n) * a_n$$

where

$$1 > a_n = (\frac{n-1}{n})^n > 0 \quad (when \quad n > 1)$$

$$\frac{\partial(\frac{n-1}{n})^n}{\partial n} = \frac{(1-\frac{1}{n})^{n-1}}{n} > 0 \quad (when \quad n > 1)$$

from which we know that the $a_n(n = 2) = \frac{1}{4}$ is the min value of a_n

According to a knowledge of $\lim_{n\to\infty}(1 + \frac{x}{n})^n = e^x$, $\lim_{n\to\infty}(1 - \frac{1}{n})^n = e^{-1}$, so the tight upper bound of a_n is e^{-1}. So a_n has a range:$[\frac{1}{4}, e^{-1})$

Since $P = (1-a_n)*a_n$ is increasing before $a_n = 1/2$ and decreasing after $a_n = 1/2$, because $1/4 < e^{-1} < 1/2$, the min value of P happens when a = 1/4, P = 3/16

$$P = (1 - a_n) * a_n \geq \frac{3}{16} \quad (for \quad n > 1)$$

Then we have the statement that the probability our sample is good is larger than some positive constant, for example, 3/16.

2 Problem II

2.1 What is the running time of this algorithm?

Outside the for loop: The algorithm spends constant time $O(1)$.
Inside the for loop: The algorithm spends $O(n)$ time.
Then the running time of the algorithm is $O(n)$.

2.2 What kind of algorithm is Randomized Approximate Median and why? What is the success probability of this algorithm?

It is a randomized algorithm.
Because although the input is fixed ,the algorithm randomly selects an item at the beginning and he output may be different. So it's is a randomized algorithm.
The success event is to uniformly select an element at random and the element neither belongs to the smallest 1/4 nor the largest 1/4 of all the elements.
So the success probability is 1/2.

2.3 How can you improve the success probability of the algorithm to over 99%? What is the running time of the new algorithm?

Pseudocode:

Algorithm 1 Function Advanced-RAM(S)

Select ($k = \log n$) items into set A = $b_1....b_{\log n} \in S$ uniformly at random
Sort them by merge-sort.
Select the (k/2)th item a_i from the set as the median element
return a_i

Success probablity(Correctness):

Randomized algorithm should show its correctness by the success probability P.
This algorithm succeeds unless the median of the k items belongs to the smallest(or the biggest) 1/4 items which equals to P_{fail}=P[more than half of the k items belongs the smallest(or the biggest) 1/4 items].
Since we know each element is selected by probability of 1/n and the probability of the selected number belonging to 1/4 part of S is 1/4:

$$
\begin{aligned}
P_{fail}/2 &= \sum_{i=k/2}^{k} P[i\ of\ k\ belongs\ to\ the\ smallest\ part] \\
&= \sum_{i=k/2}^{k} (1/4)^i * (3/4)^{k-i} * \binom{k}{i} \\
&= \sum_{i=k/2}^{k} (1/4)^i * (4/3)^i * (3/4)^k * \binom{k}{i} \\
&\leq \binom{k}{k/2} * (3/4)^k * \sum_{i=k/2}^{k} (1/3)^i \\
&\leq \binom{k}{k/2} * (3/4)^k * (1/3)^{\frac{k}{2}} * \frac{3}{2} \quad since \sum_{i=k/2}^{k} (1/3)^i = \frac{(1/3)^{\frac{k}{2}}(1 - 1/3^{\frac{k}{2}})}{1 - 1/3} \leq (1/3)^{\frac{k}{2}} * \frac{3}{2} \\
&\leq 2^{k/2} * (3/4)^k * (1/3)^{k/2} \quad since \binom{k}{k/2} = \frac{(k!)}{(\frac{k}{2}!)^2} <= 2^{k/2} \\
&= 3^{k/2} * 2^{-3k/2} \\
&\leq 2^{-k} = 2^{-\log n} = \frac{1}{n}
\end{aligned} \tag{1}
$$

So the success probability P = 1 - P_{fail} where $P_{fail} = \frac{2}{n}$
When n is very large(larger than 200), $P > 99\%$

Running time:

The algorithm takes O(k) time to select and $O(k \log k)$ time to merge-sort.
That is $O(\log n(\log \log n))$ time in total.

Space complexity:

The new assigned space is $\Theta(\log n)$
So the space complexity is $\Theta(\log n)$

3 Problem III

3.1 Suppose that in some round we have $k = \varepsilon n$ balls. At most how many balls should you expect to have in the next round?

The probability of a ball get discarded in an k-ball round is $P_f = \binom{n}{1} * \frac{1}{n} * (1 - \frac{1}{n})^{k-1} = (1 - \frac{1}{n})^{k-1}$
According to the Linearity of expectation, the expectation of the number of balls get discarded:

$$E[X] = \varepsilon n P_f = \varepsilon n (1 - \frac{1}{n})^{\varepsilon n - 1}$$

The expectation of the number of balls remained is:

$$\begin{aligned}
E[Y] = \varepsilon n - E[X] &= \varepsilon n (1 - (1 - \frac{1}{n})^{\varepsilon n - 1}) \\
&= \varepsilon n (1 - e^{-\varepsilon + \frac{1}{n}}) \quad since \lim_{n \to \infty} (1 + \frac{x}{n})^n = e^x \\
&\leq \varepsilon n (1 - e^{-\varepsilon}) \\
&\leq \varepsilon n (1 - (1 - \varepsilon)) \quad since\ 1 + x \leq e^x \\
&= \varepsilon^2 n = (\varepsilon n)^2 / n
\end{aligned} \tag{2}$$

Then the number of balls I expect to have in the next round is no more than $\varepsilon^2 n$

3.2 Assuming that everything proceeded according to expectation, prove that we would discard all the balls within O(log log n) rounds.

Assume at the beginnig of round i the balls remained is x_i,then in the round x_{i+1}:
According to the former question:

$$x_{i+1} \leq x_i^2 / n$$

Also recall the formula in the former question E[Y] = $\varepsilon n(1 - e^{-\varepsilon})$, we can see after a constant $i \geq 3$ rounds there will only be less than n/2 balls left, then:
Then in the i+t round:

$$\begin{aligned}
x_{i+t} &\leq x_{i+t-1}^2 / n \\
&\leq x_i^{2^t} / n^{2^t - 1} \\
&\leq (n/2)^{2^t} / n^{2^t - 1} \\
&\leq n / 2^{2^t}
\end{aligned} \tag{3}$$

When t = $\log \log n$, $x_{i+t} = 1$

$$t + i = constant + O(\log \log n) = O(\log \log n)$$

So it's proved that we would discard all the balls within O(log log n) rounds.

4 Problem IV

4.1 Determine the probability that a fixed person i succeeds in accessing the computer during a specific step.

$$P_i = p * (1-p)^{n-1}$$

4.2 How would you set p to maximize the above probability?

Maximizing P_i is equal to maximizing $\log P_i = \log p + (n-1)\log(1-p)$, so we need to have

$$\frac{\partial \log P_i}{\partial p} = \frac{1}{p} - (n-1)\frac{1}{1-p} = 0$$

$$p = \frac{1}{n}$$

$$P_i = \frac{1}{n} * (1 - \frac{1}{n})^{n-1}$$

4.3 For the choice of p in part(b), upper bound the probability that person i did not succeed to access the computer in any of the first t = en steps.

$P_1 = P[A\ preson\ i\ did\ not\ succeed\ to\ access\ the\ computer\ in\ a\ single\ round] = 1 - P_i$

$$\begin{aligned} P_1 &= 1 - \frac{1}{n} * (1 - \frac{1}{n})^{n-1} \\ &= 1 - \frac{1}{n} * \frac{n}{n-1} * (1 - \frac{1}{n})^{n} \\ &= 1 - \frac{1}{n-1} * e^{-1}\ since\ n \to \infty \\ &\approx 1 - \frac{1}{n} * e^{-1}\ since\ n \to \infty \\ &\leq e^{-\frac{1}{en}}\ since\ 1 + x \leq e^{x} \end{aligned} \tag{4}$$

According to chain rules: $P_t = P_1^t \leq e^{-\frac{t}{en}} = e^{-1}$

4.4 What is the number of steps t required so that the probability that person i did not succeed to access the computer in any of the first t steps is upper bounded by an inverse polynomial in n?

According to the former question, to satisfy the statement, we need $P_t = P_i^t \leq e^{-\frac{t}{en}} = O(\frac{1}{n^k})$
If $t = ken\log n$, $P_t \leq e^{-\frac{t}{en}} = e^{-\frac{ken\log n}{en}} = \frac{1}{n^k}$
So t need to satisfy $t \geq ken\log n$(k is a positive constant).

4.5 How many steps are required to guarantee that all people succeeded to access the computer with probability at least 1/n?

From former questions we know $1 - P_t =$ P[A single person successfully access the computer in t steps]
So the $P_n = (1 - P_t)^n =$ P[All people succeeded to access the computer in t steps]

$$\begin{aligned} P_n &= (1 - (1 - \frac{1}{n-1} * e^{-1})^t)^n \\ &\geq (1 - e^{-\frac{t}{en}})^n \geq \frac{1}{n} \end{aligned} \tag{5}$$

$$e^{-\frac{t}{en}} \leq 1 - n^{-\frac{1}{n}} \leq e^{-n^{-\frac{1}{n}}},\ meaning\ \frac{t}{en} \geq n^{-\frac{1}{n}}$$

$$So,\ t \geq \frac{\log(1 - n^{-\frac{1}{n}})}{1 - \frac{1}{ne}} \geq -en\ln(1 - n^{-\frac{1}{n}}) \geq en^{1-\frac{1}{n}} \approx en\ is\ required$$

5 Problem V

5.1 What is the expected time to find a good partitioning element?

Because a good partitioning element is greater than at least n/4 of the input items and smaller than at least n/4 of the input items. That means there are half of the elements can be considered as a good partition elemnt.
So the expected number of excuting the while loop is 2.
Inside the while loop the algorithm takes $O(n)$ time.
So the expected time to find a good partitioning elemnt is $O(n)$, where n = $|S|$.

5.2 What is the expected time of Randomized Quicksort-v1 on a subproblem of size $|S|$, excluding the time spent on recursive calls?

When the size $|S| \leq 3$ the excution takes $O(1)$ time.
When the size $|S|$ is larger than 3, then:
Inside the while loop: the time in (a): O(n), where n = $|S|$.
Outside the while loop: $O(1)$ time.
So in total it spends $O(n)$ time, where n = $|S|$.

5.3 We will say that a subproblem is of type j if its input consists of at most $n(\frac{3}{4})^j$ and at least $n(\frac{3}{4})^{j+1}$ items.

5.3.1 For a fixed j, how much time is spent on a subproblem of type j?

According to problem (b), the expected time spent on subproblem of type j is : $O(size) = O(n(\frac{3}{4})^j)$

5.3.2 For a fixed j, how many subproblems of type j are there?

The algorithm divide the size of problem by 1/4 to 3/4. When we consider the expected number N of subproblem of size K, it should satisfy N*K=n.
So the expected number of subprobles of type j is at most $(\frac{4}{3})^{j+1}$, that is $O((\frac{4}{3})^j)$

5.3.3 For a fixed j, how much time is spent on all subproblems of type j

The total time spent on subproblems of type j = the number of problems of type j * the expected time spent on subproblem of type j.
That is

$$O((\frac{4}{3})^j) * O(n(\frac{3}{4})^j) = O(n)$$

5.4 What is the expected running time of Randomized Quicksort-v1?

Since we have the time spent on all subproblems of type j, we only need to know the total number of subproblem types.
We know the smallest type of subproblem is 3, so: $n * (\frac{3}{4})^J = 3$,
expected number of types $J = \log_{\frac{3}{4}}(\frac{n}{3}) = O(\log n)$.
So the expected runnning time is $O(n \log n)$

6 Problem VI

6.1 Use the ideas from the previous problem to design and analyze the expected running time of a recursive randomized algorithm that returns the k-th smallest number in a set S of n distinct integers, for any k. For example, for k = dn=2e, your algorithm will return the median item.

Algorithm 2 Function kThSmallestNumber(S, k)

```
if k > |S| then
  return None
end if
if |S| ≤ 3 then
  sort S
  return the kth value
end if
for no good partitioning element has been found do
  Select an element a_i ∈ S uniformly at random
  for each element a_j ∈ S do
    Put aj in S⁻ if a_j < a_i
    Put aj in S⁺ if a_j > a_i
  end for
  if |S⁻| ≥ |S|/4 and |S⁺| ≥ |S|/4 then
    a_i is a good partitioning element
  end if
end for
if |S⁻| ≥ k then
  return kThSmallestNumber(|S⁻|, k)
else
  return kThSmallestNumber(|S⁺|,k − 1 − |S⁻|)
end if
```

Correctness:

Set the size of S $|S| = n$, then:
Base case: n = 1,2,3 The algorithm will sort and output the k th value.
Induction hypothesis: Assume that the statement is true for case $n \in [1, 2, 3....i]$ $i \geq 3$.
Inductive step: Show it true for case n=i+1:
In the algorithm, if the kth value is directly returned, the statement is correct;
otherwise after partitioning the list, this case can be converted to :
1. $n_1 > k$, Discarding the largest $(n - n_1 - 1)$ elements which do not contain the kth smalles value then finding a kth smallest value for the remained list of size $n_1 \in [1, 2, 3...i]$ $i \geq 3$, the returned value is correct according to the induction hypothesis
or
2. $n_1 < k$, Discarding the smallest (n_1) elements which do not contain the kth smalles value then finding a kth smallest value for the remained list of size $n_1 \in [1, 2, 3...i]$ $i \geq 3$, the returned value is correct according to the induction hypothesis.
So the case n=i+1 can also return the correct value.
Conclusion:
It follows that the statement is true for all n since we can apply the inductive step for n = 4; 5; 6; : : :

Running time:

For a subproblem of size n, the running time excluding the recursion step should be :
In the for loop: The expected time to find a good partition element is $O(n)$.

Outside the for loop: $O(1)$ time.
So in total is $O(n)$ time.
In a single recursion, the problem of size n is downsized into a problem of size 1/4n to 3/4n:

$$T(n) = T(3/4n) + O(n)$$

So according to the master theorem, the total running time is $O(n)$.

Space complexity:

For a subproblem of size a, the new assigned space is $\Theta(a)$;
The total recursion depth is $\log n$, the size of subproblem for depth i is at most $n * (\frac{3}{4})^i$.
So the space complexity is $\sum_{i=1}^{\log n} \Theta(n * (\frac{3}{4})^i) = \Theta(n)$