BLG335E: Analysis of Algorithms I

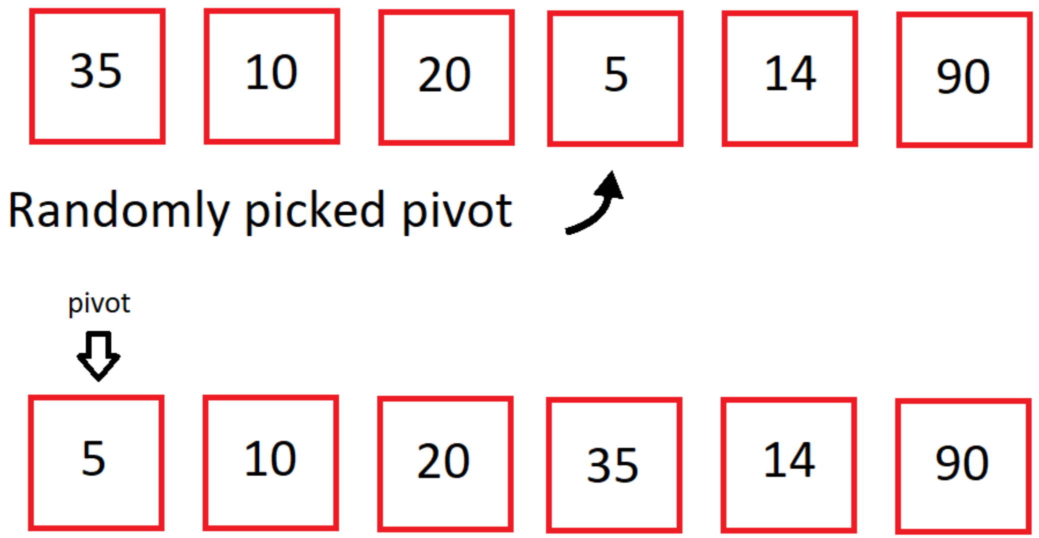
Recitation 2

Doğay Kamar – kamard@itu.edu.tr

Room:4313 - AIRLAB

Today’s Subject

- Complexity analysis of Randomized Quicksort
- Randomized QuickSort: Same as Quicksort, but the pivot in «`Partition`» function is randomly choosen.
- Then, usual «`Partition`» and «`Quicksort`» is carried out.



Randomized Quicksort

```
Algorithm RandomizedQuicksort(a, int p, int q)
{
   if (p < q) then {
      int j:= RandomizedPartition(a, p, q);
      RandomizedQuicksort(a, p, j-1);
      RandomizedQuicksort(a, j+1, q);
      }
}
```

```
Algorithm RandomizedPartition(a, int m, int p)
{
   r:=getRandomInt[m, p]
   temp:= a[m], a[m]:=a[r], a[r]:=temp
   v:=a[m]; i:=m, j:=p;
   repeat
   {
      repeat
         i:=i+1;
         until (a[i] > v);
      repeat
         j:= j-1;
         until (a[j] <= v);
      if (i>j) then t:=a[i], a[i]:=a[j]; a[j]:=t;
   } until(i >= j);
a[m] := a[j]; a[j] := v; return(j);
```

According to this pseudocode, we will find the running time of «RandomizedQuicksort».

«RandomizedPartition»

Statement	s/e	freq v = min	Freq v = max	freq v = average, a sorted	freq v = average, a unsorted	total v = min	total v = max	total v = average, a sorted	total v = average, a unsorted
r := getRandomInt[m, p]	1								
temp:=a[m], a[m]:=a[r], a[r]:= temp	3								
v:=a[m]; i:= m, j := p;	3								
repeat{	0								
repeat	0								
i:= i+1	1								
until (a[i] > v);	1								
repeat	0								
j:= j-1;	1								
until (a[j] <= v);	1								
if (i<j)	1								
then t:=a[i], a[i]:= a[j]; a[j] := t)	3								
} until(i >= j);	1								
a[m] :=a[j]; a[j] := v; return(j);	3								
Total:									

«RandomizedPartition»

Statement	s/e	freq v = min	Freq v = max	freq v = average, a sorted	freq v = average, a unsorted	total v = min	total v = max	total v = average, a sorted	total v = average, a unsorted
`r := getRandomInt[m, p]`	1	1	1	1	1	1	1	1	1
`temp:=a[m], a[m]:=a[r], a[r]:= temp`	3								
`v:=a[m]; i:= m, j := p;`	3								
`repeat{`	0								
`repeat`	0								
`i:= i+1`	1								
`until (a[i] > v);`	1								
`repeat`	0								
`j:= j-1;`	1								
`until (a[j] <= v);`	1								
`if (i<j)`	1								
`then t:=a[i], a[i]:= a[j]; a[j] := t)`	3								
`} until( i >= j);`	1								
`a[m] :=a[j]; a[j] := v; return(j);`	3								
Total:									

«RandomizedPartition»

Statement	s/e	freq v = min	Freq v = max	freq v = average, a sorted	freq v = average, a unsorted	total v = min	total v = max	total v = average, a sorted	total v = average, a unsorted
`r := getRandomInt[m, p]`	1	1	1	1	1	1	1	1	1
`temp:=a[m], a[m]:=a[r], a[r]:= temp`	3	1	1	1	1	3	3	3	3
`v:=a[m]; i:= m, j := p;`	3								
`repeat{`	0								
`repeat`	0								
`i:= i+1`	1								
`until (a[i] > v);`	1								
`repeat`	0								
`j:= j-1;`	1								
`until (a[j] <= v);`	1								
`if (i<j)`	1								
`then t:=a[i], a[i]:= a[j]; a[j] := t)`	3								
`} until( i >= j);`	1								
`a[m] :=a[j]; a[j] := v; return(j);`	3								
Total:									

«RandomizedPartition»

Statement	s/e	freq v = min	Freq v = max	freq v = average, a sorted	freq v = average, a unsorted	total v = min	total v = max	total v = average, a sorted	total v = average, a unsorted
`r := getRandomInt[m, p]`	1	1	1	1	1	1	1	1	1
`temp:=a[m], a[m]:=a[r], a[r]:= temp`	3	1	1	1	1	3	3	3	3
`v:=a[m]; i:= m, j := p;`	3	1	1	1	1	3	3	3	3
`repeat{`	0	-	-	-	-	-	-	-	-
`    repeat`	0	-	-	-	-	-	-	-	-
`        i:= i+1`	1								
`        until (a[i] > v);`	1								
`    repeat`	0								
`        j:= j-1;`	1								
`        until (a[j] <= v);`	1								
`    if (i<j)`	1								
`      then t:=a[i], a[i]:= a[j]; a[j] := t)`	3								
`   } until( i >= j);`	1								
`a[m] :=a[j]; a[j] := v; return(j);`	3								
Total:									

«RandomizedPartition»

Statement	s/e	freq v = min	Freq v = max	freq v = average, a sorted	freq v = average, a unsorted	total v = min	total v = max	total v = average, a sorted	total v = average, a unsorted
`r := getRandomInt[m, p]`	1	1	1	1	1	1	1	1	1
`temp:=a[m], a[m]:=a[r], a[r]:= temp`	3	1	1	1	1	3	3	3	3
`v:=a[m]; i:= m, j := p;`	3	1	1	1	1	3	3	3	3
`repeat{`	0	-	-	-	-	-	-	-	-
`repeat`	0	-	-	-	-	-	-	-	-
`i:= i+1`	1	1	$n-1$	$\frac{n}{2}$	$\frac{n}{2}$	1	$n-1$	$\frac{n}{2}$	$\frac{n}{2}$
`until (a[i] > v);`	1	1	$n-1$	$\frac{n}{2}$	$\frac{n}{2}$	1	$n-1$	$\frac{n}{2}$	$\frac{n}{2}$
`repeat`	0	-	-	-	-	-	-	-	-
`j:= j-1;`	1								
`until (a[j] <= v);`	1								
`if (i<j)`	1								
`then t:=a[i], a[i]:= a[j]; a[j] := t)`	3								
`} until( i >= j);`	1								
`a[m] :=a[j]; a[j] := v; return(j);`	3								
Total:									

«RandomizedPartition»

Statement	s/e	freq v = min	Freq v = max	freq v = average, a sorted	freq v = average, a unsorted	total v = min	total v = max	total v = average, a sorted	total v = average, a unsorted
`r := getRandomInt[m, p]`	1	1	1	1	1	1	1	1	1
`temp:=a[m], a[m]:=a[r], a[r]:= temp`	3	1	1	1	1	3	3	3	3
`v:=a[m]; i:= m, j := p;`	3	1	1	1	1	3	3	3	3
`repeat{`	0	-	-	-	-	-	-	-	-
`repeat`	0	-	-	-	-	-	-	-	-
`i:= i+1`	1	1	$n-1$	$\frac{n}{2}$	$\frac{n}{2}$	1	$n-1$	$\frac{n}{2}$	$\frac{n}{2}$
`until (a[i] > v);`	1	1	$n-1$	$\frac{n}{2}$	$\frac{n}{2}$	1	$n-1$	$\frac{n}{2}$	$\frac{n}{2}$
`repeat`	0	-	-	-	-	-	-	-	-
`j:= j-1;`	1	$n-1$	0	$\frac{n}{2}$	$\frac{n}{2}$	$n-1$	0	$\frac{n}{2}$	$\frac{n}{2}$
`until (a[j] <= v);`	1	$n-1$	1	$\frac{n}{2}$	$\frac{n}{2}$	$n-1$	1	$\frac{n}{2}$	$\frac{n}{2}$
`if (i<j)`	1								
`then t:=a[i], a[i]:= a[j]; a[j] := t)`	3								
`} until( i >= j);`	1								
`a[m] :=a[j]; a[j] := v; return(j);`	3								
Total:									

«RandomizedPartition»

Statement	s/e	freq v = min	Freq v = max	freq v = average, a sorted	freq v = average, a unsorted	total v = min	total v = max	total v = average, a sorted	total v = average, a unsorted
`r := getRandomInt[m, p]`	1	1	1	1	1	1	1	1	1
`temp:=a[m], a[m]:=a[r], a[r]:= temp`	3	1	1	1	1	3	3	3	3
`v:=a[m]; i:= m, j := p;`	3	1	1	1	1	3	3	3	3
`repeat{`	0	-	-	-	-	-	-	-	-
`repeat`	0	-	-	-	-	-	-	-	-
`i:= i+1`	1	1	$n-1$	$\frac{n}{2}$	$\frac{n}{2}$	1	$n-1$	$\frac{n}{2}$	$\frac{n}{2}$
`until (a[i] > v);`	1	1	$n-1$	$\frac{n}{2}$	$\frac{n}{2}$	1	$n-1$	$\frac{n}{2}$	$\frac{n}{2}$
`repeat`	0	-	-	-	-	-	-	-	-
`j:= j-1;`	1	$n-1$	0	$\frac{n}{2}$	$\frac{n}{2}$	$n-1$	0	$\frac{n}{2}$	$\frac{n}{2}$
`until (a[j] <= v);`	1	$n-1$	1	$\frac{n}{2}$	$\frac{n}{2}$	$n-1$	1	$\frac{n}{2}$	$\frac{n}{2}$
`if (i<j)`	1	1	1	1	$\frac{n}{2}$	1	1	1	$\frac{n}{2}$
`then t:=a[i], a[i]:= a[j]; a[j] := t)`	3	0	0	0	$\frac{n}{2}$	0	0	0	$\frac{3n}{2}$
`} until( i >= j);`	1								
`a[m] :=a[j]; a[j] := v; return(j);`	3								
Total:									

«RandomizedPartition»

Statement	s/e	freq v = min	Freq v = max	freq v = average, a sorted	freq v = average, a unsorted	total v = min	total v = max	total v = average, a sorted	total v = average, a unsorted
`r := getRandomInt[m, p]`	1	1	1	1	1	1	1	1	1
`temp:=a[m], a[m]:=a[r], a[r]:= temp`	3	1	1	1	1	3	3	3	3
`v:=a[m]; i:= m, j := p;`	3	1	1	1	1	3	3	3	3
`repeat{`	0	-	-	-	-	-	-	-	-
`    repeat`	0	-	-	-	-	-	-	-	-
`        i:= i+1`	1	1	$n-1$	$\frac{n}{2}$	$\frac{n}{2}$	1	$n-1$	$\frac{n}{2}$	$\frac{n}{2}$
`        until (a[i] > v);`	1	1	$n-1$	$\frac{n}{2}$	$\frac{n}{2}$	1	$n-1$	$\frac{n}{2}$	$\frac{n}{2}$
`    repeat`	0	-	-	-	-	-	-	-	-
`        j:= j-1;`	1	$n-1$	0	$\frac{n}{2}$	$\frac{n}{2}$	$n-1$	0	$\frac{n}{2}$	$\frac{n}{2}$
`        until (a[j] <= v);`	1	$n-1$	1	$\frac{n}{2}$	$\frac{n}{2}$	$n-1$	1	$\frac{n}{2}$	$\frac{n}{2}$
`    if (i<j)`	1	1	1	1	$\frac{n}{2}$	1	1	1	$\frac{n}{2}$
`      then t:=a[i], a[i]:= a[j]; a[j] := t)`	3	0	0	0	$\frac{n}{2}$	0	0	0	$\frac{3n}{2}$
`   } until( i >= j);`	1	1	1	1	$\frac{n}{2}$	1	1	1	$\frac{n}{2}$
`a[m] :=a[j]; a[j] := v; return(j);`	3	1	1	1	1	3	3	3	3
Total:						$2n + 12$	$2n + 11$	$2n + 12$	$4.5n + 10$

For all cases, running time is O(n)!

Worst-Case Analysis of Randomized Quicksort

- Different from Quicksort, we cannot select an input array of size **n** that results in worst-case, since the pivot is **randomly** picked.
- Due to this randomness, we need to compute the **expected** running time of Randomized Quicksort for any input array of size **n**.
- Since every element has equal chance to be picked, we will treat the running time as a **random variable** and try to find an upper bound for the running time.

Assume that we have an array of size **n.** Let z_i denote the i^{th} element in the **SORTED** array. And we define a random variable $Xi{,}j$ (σ) such that:

$Xi{,}j$ (σ) = 0 if z_i and z_j are not compared AND

$Xi{,}j$ (σ) = 1 if z_i and z_j are compared

Note that z_i and z_j can **not** be compared twice, since for them to be compared, one of them needs to be the **pivot** of a partition, and the pivot is not included in the next recursive partitions.

Goal: To find the expected number of comparisons in Quicksort.

Expectation definition: $E[X] = \sum_{\sigma} P(\sigma)X(\sigma) = \sum_{k} kP(x = k).$

- Note the linearity of expectation: $E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i]$

- First, let's compute the expected value of Xi,j :

$$
\begin{aligned}
E[X_{i,j}] &= P(X_{i,j} = 1) \cdot 1 + P(X_{i,j} = 0) \cdot 0 \\
&= P(X_{i,j} = 1)
\end{aligned}
$$

- $C(\sigma)$: Total number of comparisons in Quicksort given set of pivots σ:

$$C(\sigma) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{i,j}(\sigma)$$

- Our goal was to find the expected number of comparisons in Quicksort, and $E[C]$ is exactly what we want! Let's compute it.

$$E[C] = E\left[\sum_{i=1}^{n}\sum_{j=i+1}^{n} X_{i,j}(\sigma)\right]$$

$$= \sum_{i=1}^{n} E\left[\sum_{j=i+1}^{n} X_{i,j}(\sigma)\right]$$

$$= \sum_{i=1}^{n}\sum_{j=i+1}^{n} P(z_i, z_j \text{ are compared}) \quad = \sum_{i=1}^{n}\sum_{j=i+1}^{n} P(X_{i,j} = 1)$$

This two are the same thing

$$E[C] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} P(z_i, z_j \text{ are compared})$$

- Remember; at each recurrence level, elements of the array, except the pivot, are compared to pivot. So , there is only one way that $P(z_i, z_j \text{ are compared})$ is equal to one, either z_i or z_j needs to be pivot in the partition $[z_i, \ldots.., z_j]$ (Recall: z_i's are from the SORTED array). If another pivot is selected from $[z_i, \ldots.., z_j]$, z_i and z_j will be in different partitions, thus they will never be compared. So:

$$\begin{aligned} P(z_i, z_j \text{ compared}) &= P(z_i \text{ or } z_j \text{ is the first pivot picked from } [z_i, \ldots, z_j]) \\ &= \frac{1}{j-i+1} + \frac{1}{j-i+1} \\ &= \frac{2}{j-i+1} \end{aligned}$$

- On to the expected value of C:

$$E[C] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} P(z_i, z_j \text{ are compared})$$
$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

- And, if i is a fixed value, then:

$$\sum_{j=i+1}^{n} \frac{1}{j-i+1} = \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n-i+1}$$
$$\leq \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}$$

- From Harmonic numbers' upper bounds, we know that $\sum_{k=2}^{n} \frac{1}{k} \leq \ln n$. With this, we get:

$$E[C] = E\left[\sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{i,j}(\sigma)\right]$$

$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

$$\leq 2n \ln n$$

- Which shows that the expected number of comparisons in Quicksort is $2n \ln n = O(n \log n)$.

Conclusion

- Note that, we have proven that running time is $O(nlogn)$ on an **arbitrary** array, which shows that no matter what the input is, we can expect $O(nlogn)$ runtime.
- Randomized Quicksort runs in always $O(nlogn)$ as opposed to worst-case scenario of Quicksort, which runs in $O(n^2)$ for arrays sorted either in ascending or descending order.
- What about best-case scenario? Which one is better?