Randomized Algorithms

CS-6th

Instructor: Dr. Ayesha Enayet

- These are algorithms that make use of randomness in their computation/logic.
- Random selection ensures that outcomes are not solely determined by the external inputs of the problem.
- By introducing randomness, we can avoid worst-case scenarios.
- Can give a better **expected time** complexity.

Quick Sort (Deterministic)

i=j-1	j				Pivot
index	1	2	3	4	5
Elements	2	8	3	5	7

1. Compare pivot with the element at j:
   - If j>pivot, increment j
   - If j<pivot, increment i swap the values of i and j

   <repeat till j==pivot>
2. Increment i and swap i and pivot

Quick Sort (Deterministic)

j<pivot

i=j-1=0	j				Pivot
index	1	2	3	4	5
Elements	2	8	3	5	7

1. Compare pivot with the element at j:
   - If j>pivot, increment j
   - If j<pivot, increment i swap the values of i and j

   <repeat till j==pivot>
2. Increment i and swap i and pivot

Quick Sort (Deterministic)

j<pivot

	i,j=1				Pivot
index	1	2	3	4	5
Elements	2	8	3	5	7

1. Compare pivot with the element at j:
   - If j>pivot, increment j
   - If j<pivot, increment i swap the values of i and j
     - Increment j

   <repeat till j==pivot>
2. Increment i and swap i and pivot

Quick Sort (Deterministic)

j>pivot

	i	j			Pivot
index	1	2	3	4	5
Elements	2	8	3	5	7

1. Compare pivot with the element at j:
   - If j>pivot, increment j
   - If j<pivot, increment i swap the values of i and j
     - Increment j

   <repeat till j==pivot>
2. Increment i and swap i and pivot

Quick Sort (Deterministic)

j<pivot

	i		j		Pivot
index	1	2	3	4	5
Elements	2	8	3	5	7

1. Compare pivot with the element at j:
   - If j>pivot, increment j
   - If j<pivot, increment i swap the values of i and j
     - Increment j

   <repeat till j==pivot>
2. Increment i and swap i and pivot

Quick Sort (Deterministic)

		i	j		Pivot
index	1	2	3	4	5
Elements	2	3	8	5	7

1. Compare pivot with the element at j:
   - If j>pivot, increment j
   - If j<pivot, increment i swap the values of i and j
     - Increment j

   <repeat till j==pivot>
2. Increment i and swap i and pivot

Quick Sort (Deterministic)

j<pivot

		i		j	Pivot
index	1	2	3	4	5
Elements	2	3	8	5	7

1. Compare pivot with the element at j:
    - If j>pivot, increment j
    - If j<pivot, increment i swap the values of i and j
        - Increment j

   <repeat till j==pivot>
2. Increment i and swap i and pivot

Quick Sort (Deterministic)

			i	j	Pivot
index	1	2	3	4	5
Elements	2	3	5	8	7

1. Compare pivot with the element at j:
   - If j>pivot, increment j
   - If j<pivot, increment i swap the values of i and j
     - Increment j

   <repeat till j==pivot>
2. Increment i and swap i and pivot

Quick Sort (Deterministic)

			i		j,Pivot
index	1	2	3	4	5
Elements	2	3	5	8	7

1. Compare pivot with the element at j:
   - If j>pivot, increment j
   - If j<pivot, increment i swap the values of i and j
     - Increment j

   <repeat till j==pivot>
2. Increment i and swap i and pivot

Quick Sort (Deterministic)

				i	j,Pivot
index	1	2	3	4	5
Elements	2	3	5	7	8

1. Compare pivot with the element at j:
   - If j>pivot, increment j
   - If j<pivot, increment i swap the values of i and j
     - Increment j

   <repeat till j==pivot>
2. Increment i and swap i and pivot

Quick Sort (Deterministic)

				Pivot	
index	1	2	3	4	5
Elements	2	3	5	7	8

1. Compare pivot with the element at j:
   - If j>pivot, increment j
   - If j<pivot, increment i swap the values of i and j
     - Increment j

   <repeat till j==pivot>
2. Increment i and swap i and pivot

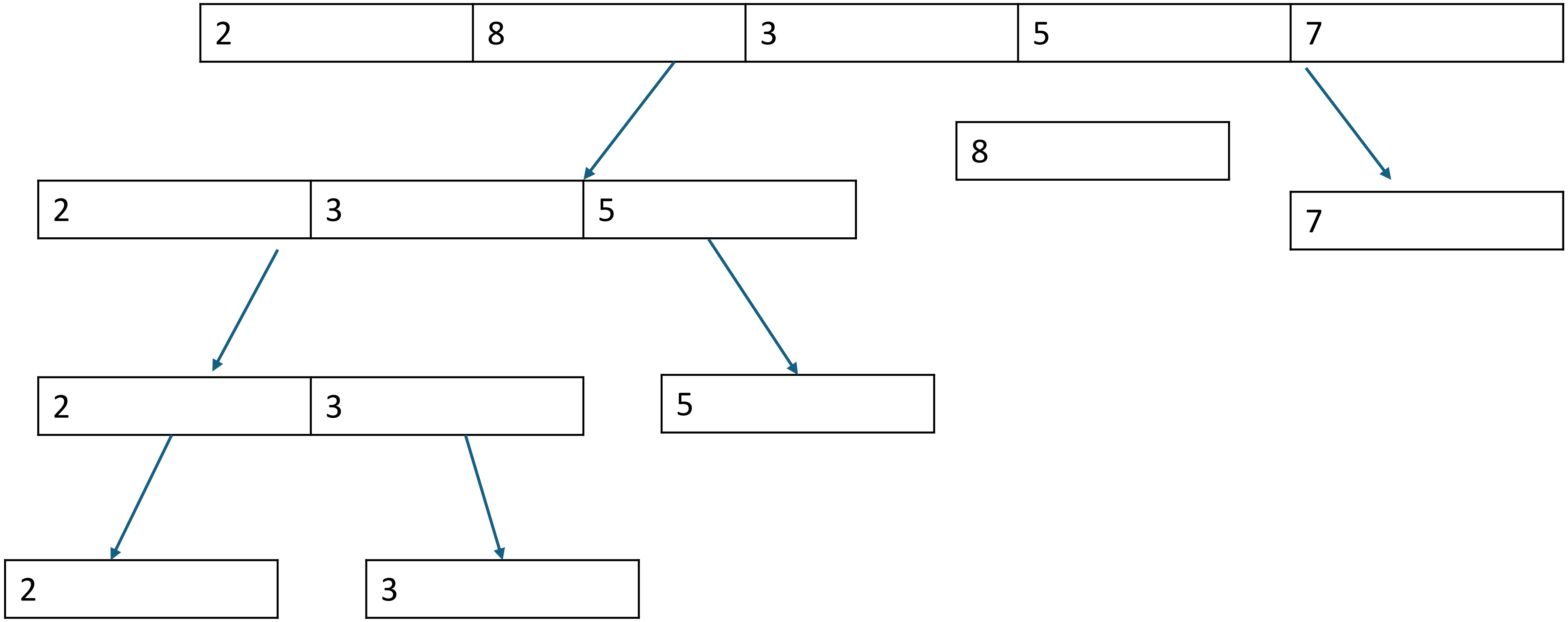
Recursive Divide-And-Conquer

				Pivot	
index	1	2	3	4	5
Elements	2	3	5	7	8

Pivot
4
7

j		Pivot
1	2	3
2	3	5

j,pivot
5
8


2
8
3
5
7
8
7
2
3
5
2
3
5
2
3

Worst case Time Complexity?

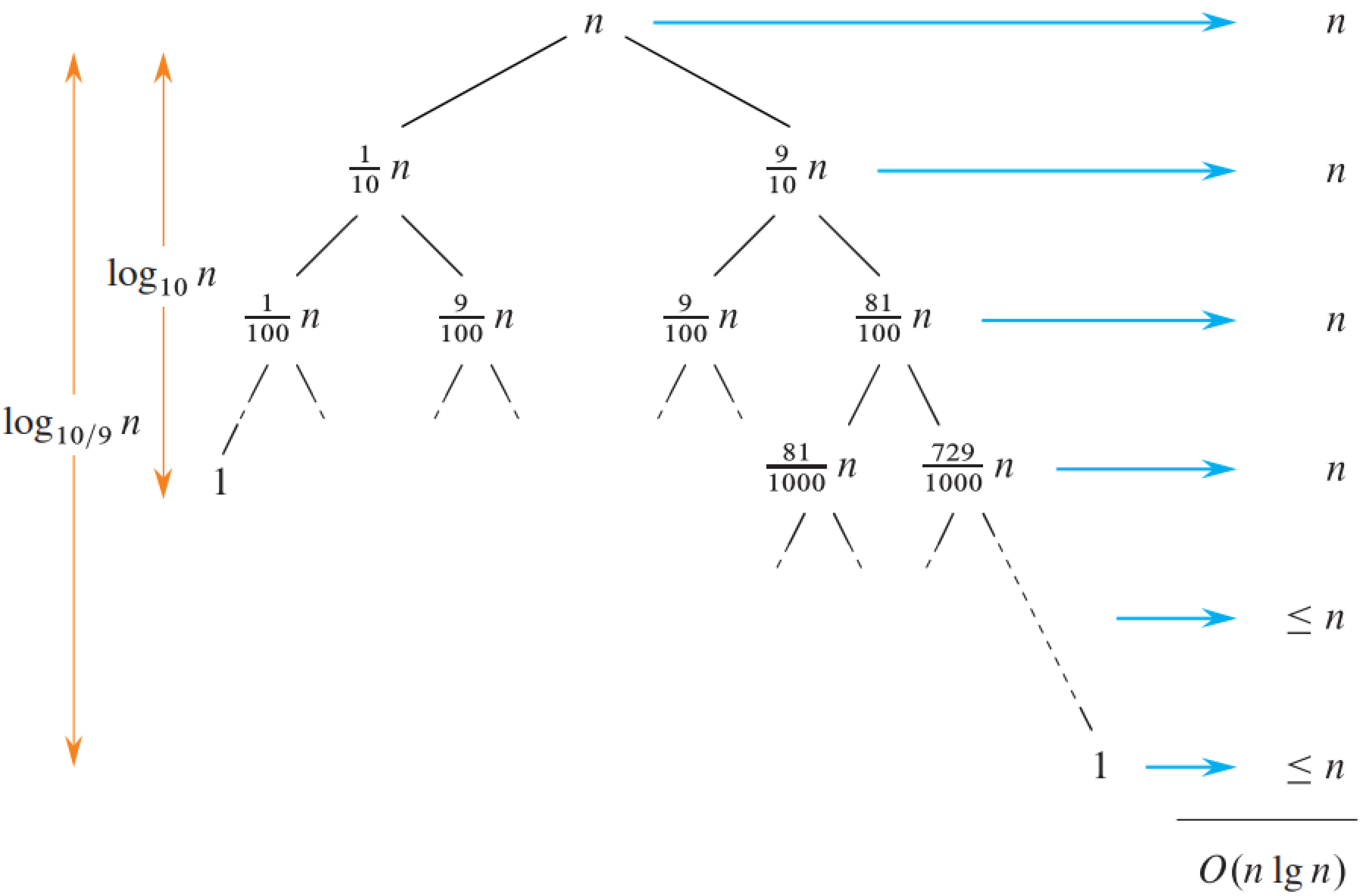
$$T(n)=T(n-1)+n=O(n^2)$$

Best case Time Complexity?

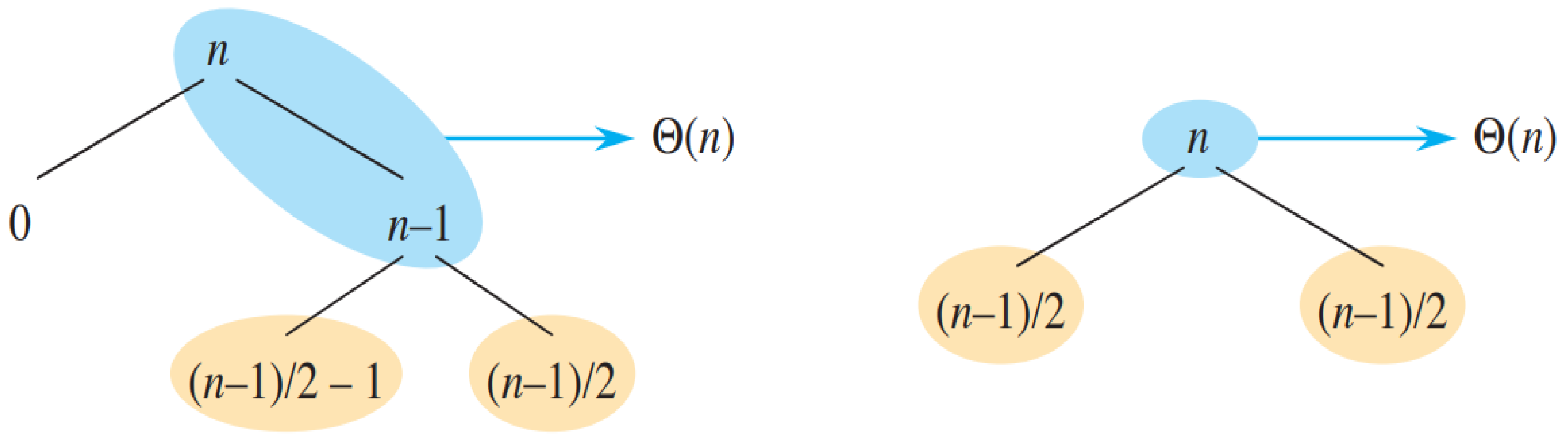
Balanced Partitioning

$$T(n)=2T(n/2)+O(n)=O(n\lg n)$$

Average case Time Complexity?


n
1/10 n
9/10 n
1/100 n
9/100 n
9/100 n
81/100 n
81/1000 n
729/1000 n
1
1
log_10 n
log_10/9 n
n
n
n
n
≤ n
≤ n
O(n lg n)

- In the average case, PARTITION produces a mix of “good” and “bad” splits.
- In a recursion tree for an average-case execution of PARTITION, the good and bad splits are distributed randomly throughout the tree.



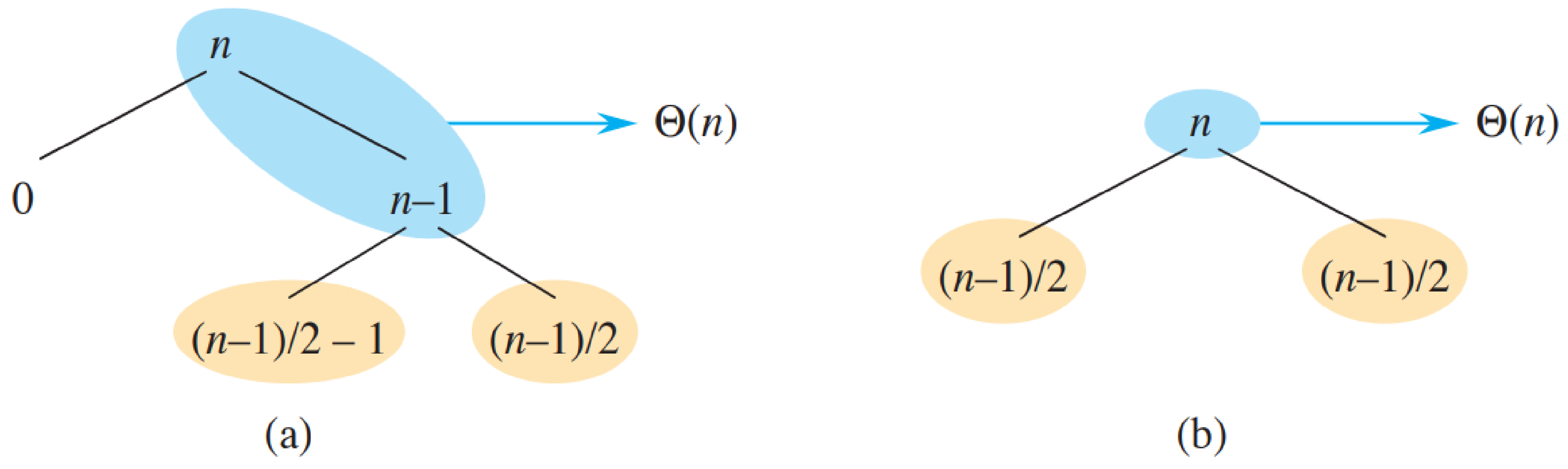


Figure 7.5 **(a)** Two levels of a recursion tree for quicksort. The partitioning at the root costs n and produces a "bad" split: two subarrays of sizes 0 and $n - 1$. The partitioning of the subarray of size $n - 1$ costs $n - 1$ and produces a "good" split: subarrays of size $(n - 1)/2 - 1$ and $(n - 1)/2$. **(b)** A single level of a recursion tree that is well balanced. In both parts, the partitioning cost for the subproblems shown with blue shading is $\Theta(n)$. Yet the subproblems remaining to be solved in (a), shown with tan shading, are no larger than the corresponding subproblems remaining to be solved in (b).

- We assume that all the permutations of the input are equally likely.
- Thus, the running time of quicksort, when levels alternate between good and bad splits, is like the running time for good splits alone: still O(n lg n), but with a slightly larger constant hidden by the O-notation.

```
QUICKSORT(A, p, r)
  if p < r
      // Partition the subarray around the pivot, which ends up in A[q].
      q = PARTITION(A, p, r)
      QUICKSORT(A, p, q - 1)   // recursively sort the low side
      QUICKSORT(A, q + 1, r)   // recursively sort the high side
```

```
PARTITION(A, p, r)
x = A[r]                                  // the pivot
i = p - 1                                 // highest index into the low side
for j = p to r - 1                        // process each element other than the pivot
    if A[j] ≤ x                           // does this element belong on the low side?
        i = i + 1                         // index of a new slot in the low side
        exchange A[i] with A[j]           // put this element there
exchange A[i + 1] with A[r]               // pivot goes just to the right of the low side
return i + 1                              // new index of the pivot
```

Randomized Quick Sort

```
RANDOMIZED-PARTITION(A, p, r)
i = RANDOM(p, r)
exchange A[r] with A[i]
return PARTITION(A, p, r)

RANDOMIZED-QUICKSORT(A, p, r)
if p < r
    q = RANDOMIZED-PARTITION(A, p, r)
    RANDOMIZED-QUICKSORT(A, p, q - 1)
    RANDOMIZED-QUICKSORT(A, q + 1, r)
```

Randomized Quick Sort

- We want to reduce the probability of the occurrence of the worst case by introducing randomization.
- Judicious randomization can sometimes be added to an algorithm to obtain good expected performance over all inputs.
- The pivot is chosen randomly, we expect the split of the input array to be reasonably well balanced on average.
- For quicksort, randomization yields a fast and practical algorithm.

Expected running time

The expected running time of RANDOMIZED-QUICKSORT on an input of n distinct elements is O(n lg n).

Proof

- Let the n distinct elements be $z_1 < z_2 < \ldots < z_n$, and for 1<=i < j<= n, define the indicator random variable X_{ij}=1{ iff z_i is compared to z_j}. Each pair is compared at most once, and so we can express X as follows:

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} \; .$$

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} \ .$$

Taking the expectation on both sides.

Informally, the expected value is the arithmetic mean of the possible values a random variable can take, weighted by the probability of those outcomes (weighted average).

$$\mathrm{E}[X] = \mathrm{E}\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right]$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathrm{E}[X_{ij}] \qquad \text{(by linearity of expectation)}$$

the expected value of a sum of random variables is the sum of the expected values of the variables

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared with } z_j\} \quad \text{(by Lemma 5.1)}$$

Lemma 5.1

Given a sample space S and an event A in the sample space S, let $X_A = \mathrm{I}\{A\}$. Then $\mathrm{E}[X_A] = \Pr\{A\}$.

Proof By the definition of an indicator random variable from equation (5.1) and the definition of expected value, we have

$$\begin{aligned} \mathrm{E}[X_A] &= \mathrm{E}[\mathrm{I}\{A\}] \\ &= 1 \cdot \Pr\{A\} + 0 \cdot \Pr\{\overline{A}\} \\ &= \Pr\{A\} , \end{aligned}$$

Lemma 7.3

$$
\begin{aligned}
\Pr\{z_i \text{ is compared with } z_j\} &= \Pr\{z_i \text{ or } z_j \text{ is the first pivot chosen from } Z_{ij}\} \\
&= \Pr\{z_i \text{ is the first pivot chosen from } Z_{ij}\} \\
&\quad + \Pr\{z_j \text{ is the first pivot chosen from } Z_{ij}\} \\
&= \frac{2}{j-i+1},
\end{aligned}
$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared with } z_j\} \quad \text{(by Lemma 5.1)}$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \quad \text{(by Lemma 7.3)} .$$

$$\mathrm{E}[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

$$\begin{aligned}
\mathrm{E}[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \\
&< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k}
\end{aligned}$$

$$
\begin{aligned}
&< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} \\
&= \sum_{i=1}^{n-1} O(\lg n) \\
&= O(n \lg n) \, .
\end{aligned}
$$

By Harmonic series

Quick Select

- Input: A list of numbers S; an integer k
- Output: The kth smallest element of S

- Here's a divide-and-conquer approach to selection. For any number v, imagine splitting list S into three categories: elements smaller than v, those equal to v (there might be duplicates), and those greater than v. Call these S_L, S_v, and S_R respectively. For instance, if the array

S:

2	36	5	21	8	13	11	20	5	4	1

is split on $v = 5$, the three subarrays generated are

S_L:

2	4	1

S_v:

5	5

S_R:

36	21	8	13	11	20

The three cases

$$\text{selection}(S,k) = \begin{cases} \text{selection}(S_L,k) & \text{if } k \le |S_L| \\ v & \text{if } |S_L| < k \le |S_L| + |S_v| \\ \text{selection}(S_R, k - |S_L| - |S_v|) & \text{if } k > |S_L| + |S_v|. \end{cases}$$

Time Complexity analysis

- Worst Case: $O(n^2)$
- Best Case: $T(n)=T(n/2)+ O(n)=O(n)$

Average Case

- To distinguish between lucky and unlucky choices of v, we will call v good if it lies within the **25th to 75th** percentile of the array that it is chosen from.

Average Case

- Given that a randomly chosen v has a 50% chance of being good, how many v's do we need to pick on average before getting a good one?
- **Lemma** On average a fair coin needs to be tossed two times before a "heads" is seen.
- E=1+$\frac{1}{2}E = 2$

Average Case

- Therefore, after two split operations on average, the array will shrink to at most three fourths of its size. Letting T(n) be the expected running time on an array of size n, we get:

$$T(n) \leq T(3n/4) + O(n).$$

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

$$\begin{aligned}&\text{Time taken on an array of size } n\\ &\quad\leq \text{(time taken on an array of size } 3n/4) + \text{(time to reduce array size to} \leq 3n/4),\end{aligned}$$