CO202 – Software Engineering – Algorithms

Randomised Algorithms - Solutions

Exercise 1: Illustrate the Operations of Partition

```
PARTITION(A,p,r)
 1: x = A[r]
 2: i = p-1
 3: for j = p to r-1
 4:     if A[j] ≤ x
 5:         i = i+1
 6:         SWAP(A[i],A[j])
 7: SWAP(A[i+1],A[r])
 8: return i+1
```

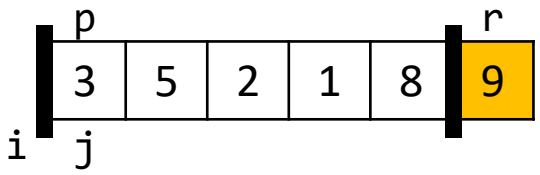
$$A = \langle 3, 5, 2, 1, 8, 9 \rangle$$

Exercise 1: Illustrate the Operations of Partition

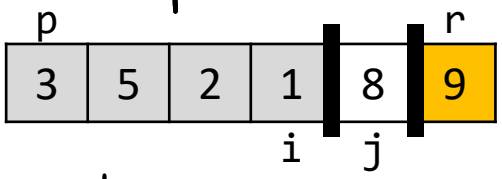
```
PARTITION(A,p,r)
 1: x = A[r]
 2: i = p-1
 3: for j = p to r-1
 4:     if A[j] ≤ x
 5:         i = i+1
 6:         SWAP(A[i],A[j])
 7: SWAP(A[i+1],A[r])
 8: return i+1
```

$$A = \langle 3, 5, 2, 1, 8, 9 \rangle$$

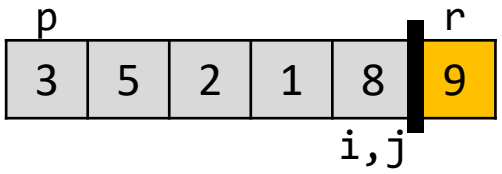
start of for-loop



end of for-loop



before return



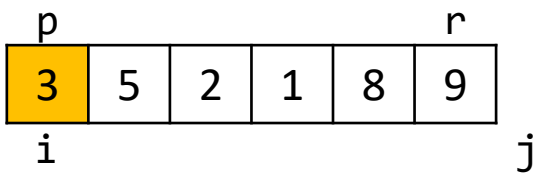
Exercise 2: The Original Partition Algorithm

$$A = \langle 3, 5, 2, 1, 8, 9 \rangle$$

```
HOARE-PARTITION(A,p,r)
 1: x = A[p]
 2: i = p
 3: j = r+1
 4: while TRUE
 5:     repeat
 6:         j = j-1
 7:     until A[j] ≤ x or j == p
 8:     repeat
 9:         i = i+1
10:     until A[i] ≥ x or i == r
11:     if i < j
12:         SWAP(A[i],A[j])
13:     else
14:         SWAP(A[p],A[j])
15:         return j
```

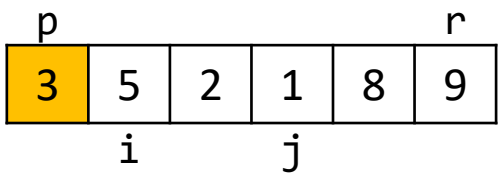
Exercise 2: The Original Partition Algorithm

```
HOARE-PARTITION(A,p,r)
 1: x = A[p]
 2: i = p
 3: j = r+1
 4: while TRUE
 5:     repeat
 6:         j = j-1
 7:     until A[j] ≤ x or j == p
 8:     repeat
 9:         i = i+1
10:     until A[i] ≥ x or i == r
11:     if i < j
12:         SWAP(A[i],A[j])
13:     else
14:         SWAP(A[p],A[j])
15:         return j
```



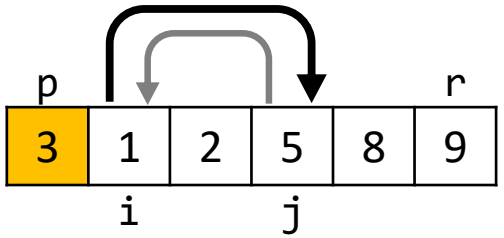
Exercise 2: The Original Partition Algorithm

```
HOARE-PARTITION(A,p,r)
 1: x = A[p]
 2: i = p
 3: j = r+1
 4: while TRUE
 5:     repeat
 6:         j = j-1
 7:     until A[j] ≤ x or j == p
 8:     repeat
 9:         i = i+1
10:     until A[i] ≥ x or i == r
11:     if i < j
12:         SWAP(A[i],A[j])
13:     else
14:         SWAP(A[p],A[j])
15:         return j
```



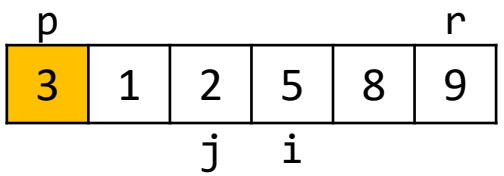
Exercise 2: The Original Partition Algorithm

```
HOARE-PARTITION(A,p,r)
 1: x = A[p]
 2: i = p
 3: j = r+1
 4: while TRUE
 5:     repeat
 6:         j = j-1
 7:     until A[j] ≤ x or j == p
 8:     repeat
 9:         i = i+1
10:     until A[i] ≥ x or i == r
11:     if i < j
12:         SWAP(A[i],A[j])
13:     else
14:         SWAP(A[p],A[j])
15:         return j
```



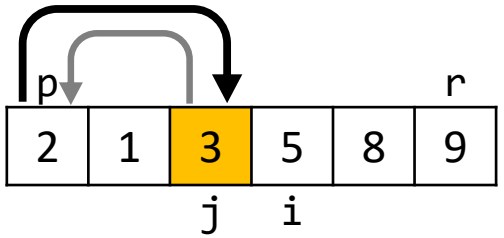
Exercise 2: The Original Partition Algorithm

```
HOARE-PARTITION(A,p,r)
 1: x = A[p]
 2: i = p
 3: j = r+1
 4: while TRUE
 5:     repeat
 6:         j = j-1
 7:     until A[j] ≤ x or j == p
 8:     repeat
 9:         i = i+1
10:     until A[i] ≥ x or i == r
11:     if i < j
12:         SWAP(A[i],A[j])
13:     else
14:         SWAP(A[p],A[j])
15:         return j
```



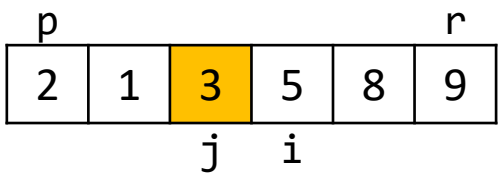
Exercise 2: The Original Partition Algorithm

```
HOARE-PARTITION(A,p,r)
 1: x = A[p]
 2: i = p
 3: j = r+1
 4: while TRUE
 5:     repeat
 6:         j = j-1
 7:     until A[j] ≤ x or j == p
 8:     repeat
 9:         i = i+1
10:     until A[i] ≥ x or i == r
11:     if i < j
12:         SWAP(A[i],A[j])
13:     else
14:         SWAP(A[p],A[j])
15:         return j
```



Exercise 2: The Original Partition Algorithm

```
HOARE-PARTITION(A,p,r)
 1: x = A[p]
 2: i = p
 3: j = r+1
 4: while TRUE
 5:     repeat
 6:         j = j-1
 7:     until A[j] ≤ x or j == p
 8:     repeat
 9:         i = i+1
10:     until A[i] ≥ x or i == r
11:     if i < j
12:         SWAP(A[i],A[j])
13:     else
14:         SWAP(A[p],A[j])
15:         return j
```



Exercise 3: Randomised BST Insert

$\langle 2, 3, 1, 5, 7, 8, 9 \rangle$ $\langle 0, 1, 0, 1, 1, 0, 0 \rangle$

```
INSERT-RAND(t,z)
 1: if t == NIL
 2:     return z
 3: r = RANDOM(1,t.size+1)
 4: if r == 1
 5:     return ROOT-INSERT(t,z)
 6: if z.key < t.key
 7:     t.left = INSERT-RAND(t.left,z)
 8: else
 9:     t.right = INSERT-RAND(t.right,z)
10: t.size = t.size + 1
11: return t
```

```
ROOT-INSERT(t,z)
 1: if t == NIL
 2:     return z
 3: if z.key < t.key
 4:     t.left = ROOT-INSERT(t.left,z)
 5:     t.size = t.size + 1
 6:     return RIGHT-ROTATE(t)
 7: else
 8:     t.right = ROOT-INSERT(t.right,z)
 9:     t.size = t.size + 1
10:     return LEFT-ROTATE(t)
```

```
LEFT-ROTATE(t)
 1: r = t.right
 2: t.right = r.left
 3: r.left = t
 4: r.size = t.size
 5: t.size -= r.right.size + 1
 6: return r
```

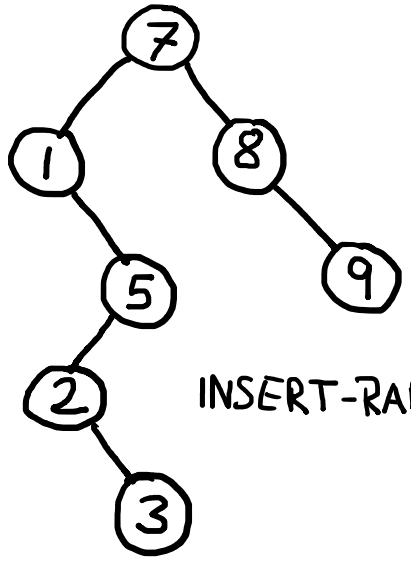
```
RIGHT-ROTATE(t)
 1: l = t.left
 2: t.left = l.right
 3: l.right = t
 4: l.size = t.size
 5: t.size -= l.left.size + 1
 6: return l
```

Exercise 3: Randomised BST Insert

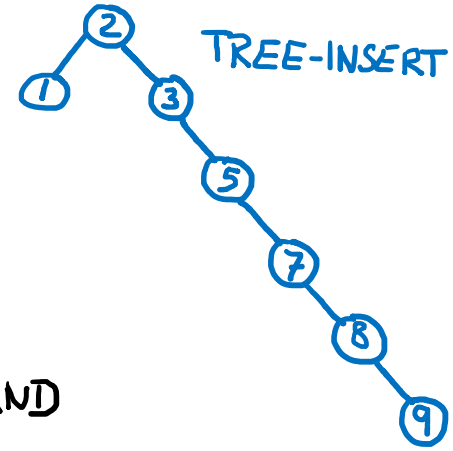
```
INSERT-RAND(t,z)
 1: if t == NIL
 2:     return z
 3: r = RANDOM(1,t.size+1)
 4: if r == 1
 5:     return ROOT-INSERT(t,z)
 6: if z.key < t.key
 7:     t.left = INSERT-RAND(t.left,z)
 8: else
 9:     t.right = INSERT-RAND(t.right,z)
10: t.size = t.size + 1
11: return t
```

```
ROOT-INSERT(t,z)
 1: if t == NIL
 2:     return z
 3: if z.key < t.key
 4:     t.left = ROOT-INSERT(t.left,z)
 5:     t.size = t.size + 1
 6:     return RIGHT-ROTATE(t)
 7: else
 8:     t.right = ROOT-INSERT(t.right,z)
 9:     t.size = t.size + 1
10:     return LEFT-ROTATE(t)
```

$\langle 2, 3, 1, 5, 7, 8, 9 \rangle$ $\langle 0, 1, 0, 1, 1, 0, 0 \rangle$



INSERT-RAND



TREE-INSERT

```
LEFT-ROTATE(t)
 1: r = t.right
 2: t.right = r.left
 3: r.left = t
 4: r.size = t.size
 5: t.size -= r.right.size + 1
 6: return r
```

```
RIGHT-ROTATE(t)
 1: l = t.left
 2: t.left = l.right
 3: l.right = t
 4: l.size = t.size
 5: t.size -= l.left.size + 1
 6: return l
```

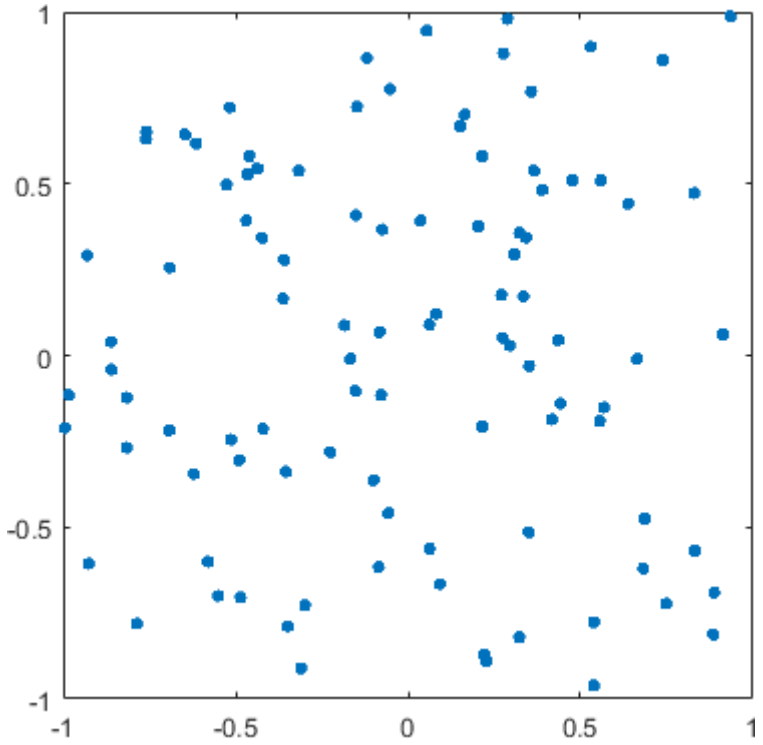
Exercise 4: Approximating Pi

How can we approximate pi using random numbers?

Exercise 4: Approximating Pi

How can we approximate pi using random numbers?

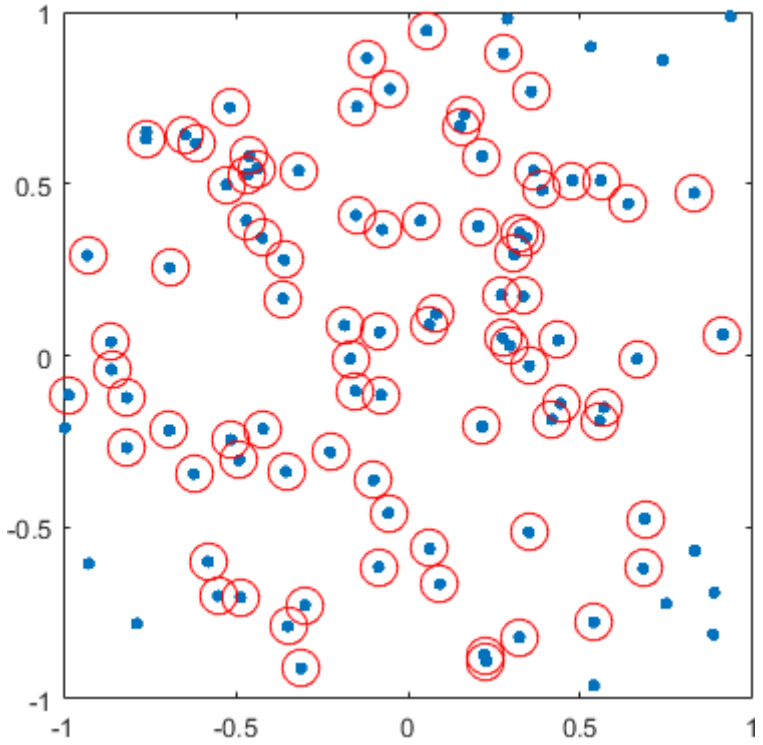
- Generate n random points in a $[-1,1] \times [-1,1]$ square
- Define m as the number of points within distance 1 from $(0,0)$



Exercise 4: Approximating Pi

How can we approximate pi using random numbers?

- Generate n random points in a $[-1,1] \times [-1,1]$ square
- Define m as the number of points within distance 1 from $(0,0)$
- The ratio $m/n \approx \pi r^2/4r^2$ with $r = 1$, $\pi \approx 4m/n$



$$n = 100$$

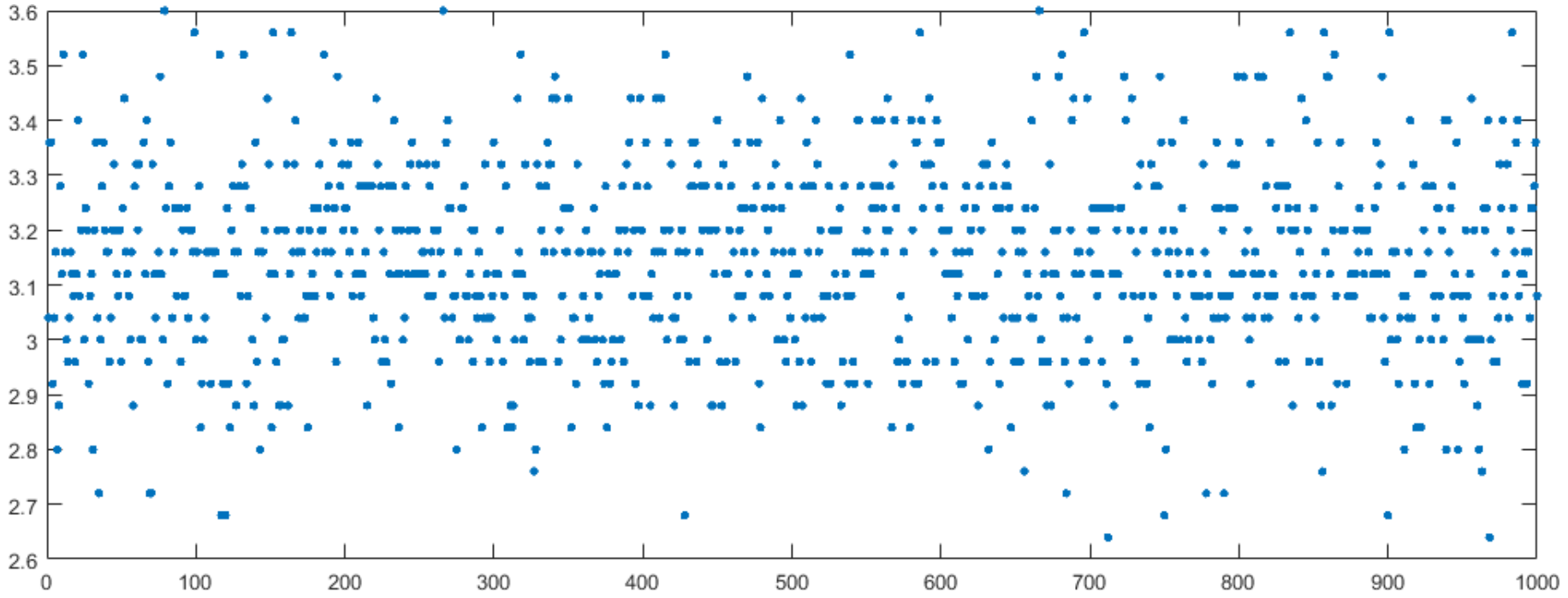
$$m = 87$$

$$\pi \approx 4 \times \frac{87}{100} = 3.48$$

Exercise 4: Approximating Pi

How can we approximate pi using random numbers?

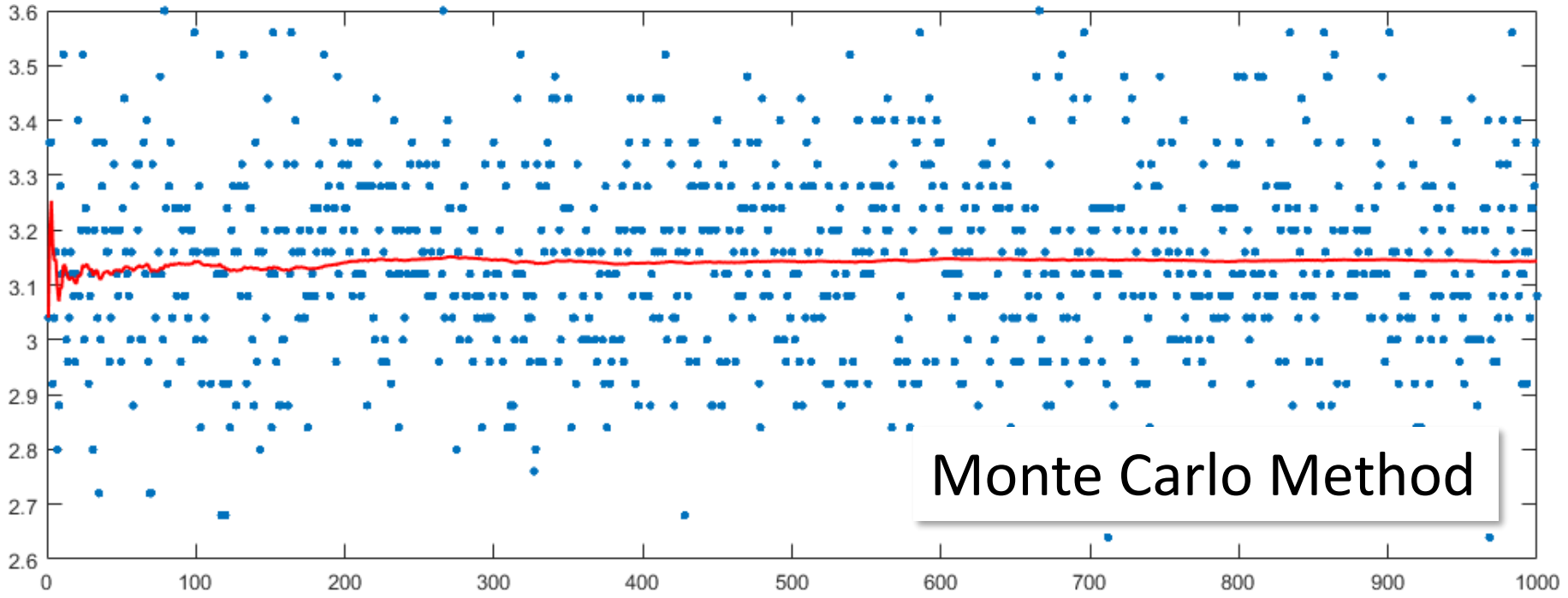
- Generate n random points in a $[-1,1] \times [-1,1]$ square
- Define m as the number of points within distance 1 from $(0,0)$
- The ratio $m/n \approx \pi r^2/4r^2$ with $r = 1$, $\pi \approx 4m/n$



Exercise 4: Approximating Pi

How can we approximate pi using random numbers?

- Generate n random points in a $[-1,1] \times [-1,1]$ square
- Define m as the number of points within distance 1 from $(0,0)$
- The ratio $m/n \approx \pi r^2/4r^2$ with $r = 1$, $\pi \approx 4m/n$



Exercise 5: Finding the k-th Smallest Element

Given set $A = \{a_1, \ldots, a_n\}$, find the k-th smallest Element

Exercise 5: Finding the k-th Smallest Element

Given set $A = \{a_1, \ldots, a_n\}$, find the k-th smallest Element

Solution 1: Sort the sequence, return the k-th element

Running time complexity $\Theta(n \log n)$

Exercise 5: Finding the k-th Smallest Element

Given set $A = \{a_1, \ldots, a_n\}$, find the k-th smallest Element

Solution 2: Randomised algorithm

Exercise 5: Finding the k-th Smallest Element

Given set $A = \{a_1, \dots, a_n\}$, find the k-th smallest Element

Solution 2: Randomised algorithm

Algorithm $FKS(A, k)$

Step 1: If $n = 1$, output a_1, otherwise choose random $i \in \{1, \dots, n\}$

Step 2: Compute $A_< = \{b \in A | b < a_i\}$ and $A_> = \{c \in A | c > a_i\}$

Step 3:

- if $|A_<| > k$, then call $FKS(A_<, k)$
- if $|A_<| = k - 1$, then output a_i
- otherwise call $FKS(A_>, k - |A_<| - 1)$