Data Structures and Algorithms

(ESO207)

Lecture 33

- Algorithm for ith order statistic of a set S.

Problem definition

Given a set S of n elements
compute ith **smallest** element from S.

Applications:

Trivial algorithm:

But sorting takes O(n log n) time and appears to be an overkill for this simple problem.

AIM: To design an algorithm with **O(n)** time complexity.

Assumption (For the sake of **neat description** and **analysis** of algorithms of this lecture):

- All elements of S are assumed to be **distinct**.

A motivational background

Though it was **intuitively appealing** to believe that there exists an O(n) time algorithm to compute ith smallest element, it remained a challenge for many years to design such an algorithm…

In **1972**, five well known researchers: **Blum, Floyd, Pratt, Rivest**, and **Tarjan** designed the **O**(n) time algorithm. It was designed during a **lunch break** of a conference when these five researchers sat together for the first time to solve the problem.

In this way, the problem which remained unsolved for many years got solved in less than an hour. But one should not ignore the efforts these researchers spent for years before arriving at the solution … It was their effort whose fruit got ripened in that hour ☺.

Notations

We shall now introduce some notations which will help in a neat description of the algorithm.

Notations

- S **:**

  the given set of n elements.
- e_i :

  ith **smallest** element of S.
- $S_{<x}$ **:**

  subset of S consisting of all elements **smaller than** x**.**
- $S_{>x}$ **:**

  subset of S consisting of all elements **greater than** x**.**
- **rank(**S**,**x**) :**

  1 + number of elements in S that are smaller than x**.**
- **Partition(**S**,**x**):**

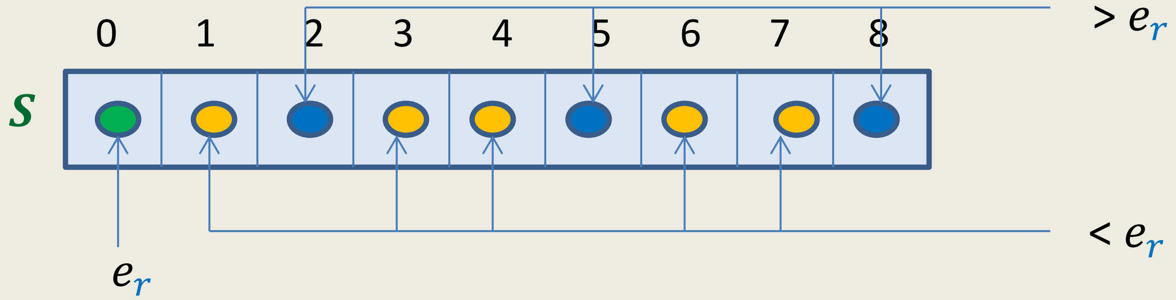
  algorithm to partition S into $S_{<x}$ and $S_{>x}$**;**

  this algorithm returns **(**$S_{<x}$**,** $S_{>x}$**,** r**)** where r**=rank(**S**,**x**).**

Why should such an algorithm exist ?

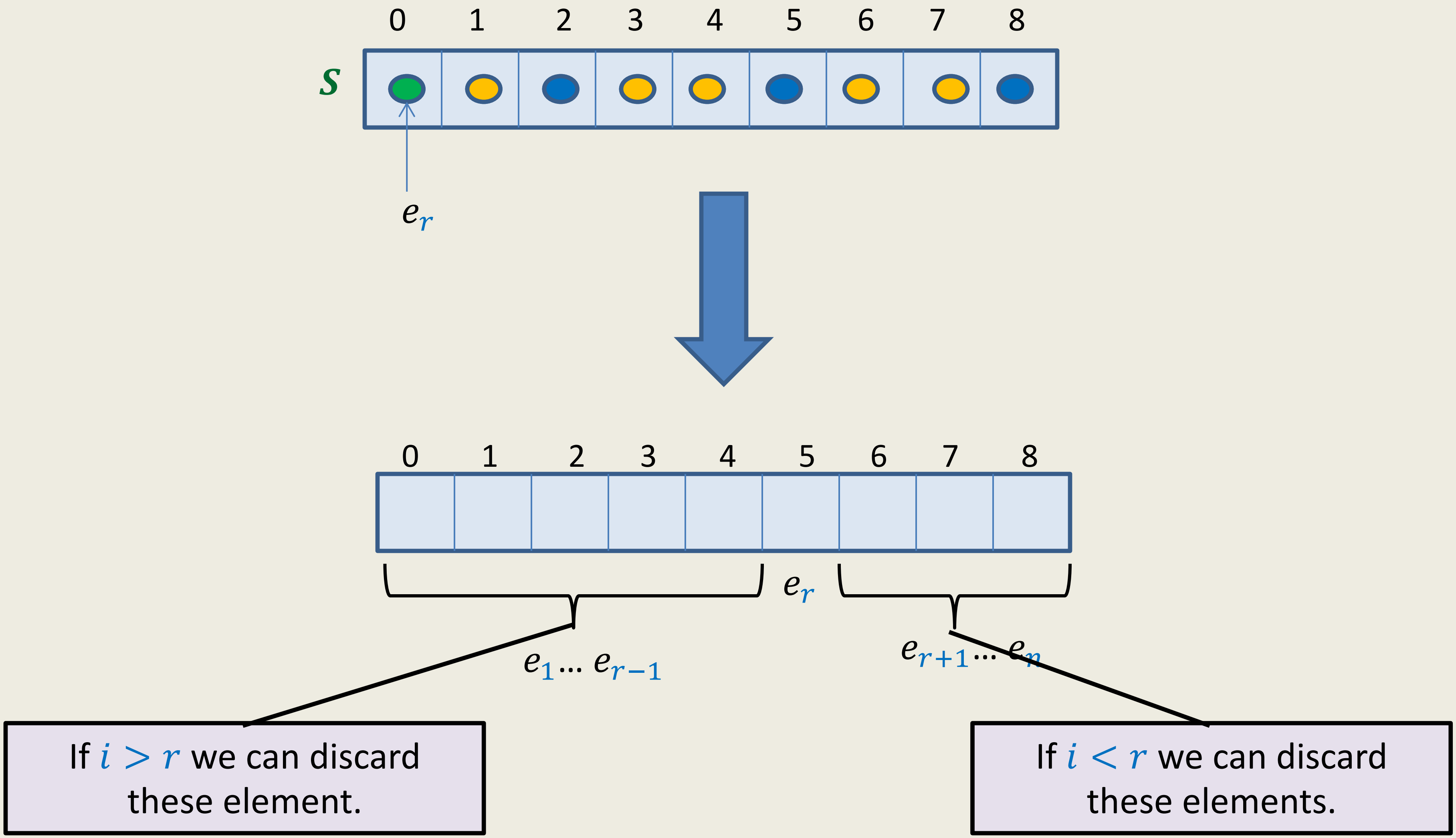
(inspiration from **QuickSort**)

QuickSelect(S,i)



What happens during
Partition(S,0,8)

QuickSelect(S,i)



Pseudocode for QuickSelect(S,i)

QuickSelect(S,i)
{ Pick an element x from S;
  ($S_{<x}$, $S_{>x}$, r)← Partition(S,x);
  If(i = r) return x;
  Else If (i< r)
      QuickSelect($S_{<x}$,i)
    Else
      QuickSelect($S_{>x}$,$i - r$);
}

Average case time complexity: O(n)

Worst case time complexity : O(n^2)

Analysis is simpler than **Quick Sort**.

Towards worst case O(n) time algorithm …

Key ideas

- **Inspiration** from some recurrences.

isn’t is surprising that knowledge of recurrence can help in the design an efficient algorithm) ☺

- Concept of **approximate median**

This is the usual trick:
When a problem appears difficult, weaken the problem and try to solve it.

- Learning from **QuickSelect**($\boldsymbol{S}$,$\boldsymbol{i}$)

This is also a natural choice.
Can we fine tune this algorithm to achieve our goal ?

Learning from recurrences

Question: what is the solution of recurrence T(n) = cn + T($9n/10$) ?

Answer: O(n).

Sketch (by gradual unfolding):

T(n) = cn + c $9n/10$ + c $81\ n/100$ + …

= $cn[1$ + $9/10$ + $81/100$ + …$]$

= O(n)

Lesson 1 :

Solution for T(n) = cn + T(an) is O(n) if $0 < a < 1$.

Learning from recurrences

Question: what is the solution of recurrence

$$T(n) = cn + T(n/6) + T(5n/7) \ ?$$

Answer: O(n).

Sketch: (by induction)

Assertion: $T(n) \le c_1 n$.

Induction step: $T(n) = cn + T(n/6) + T(5n/7)$

$$\le cn + \frac{37}{42} c_1 n$$

$$\le c_1 n \quad \text{if } c_1 \ge \frac{42}{5} c$$

Lesson 2 :

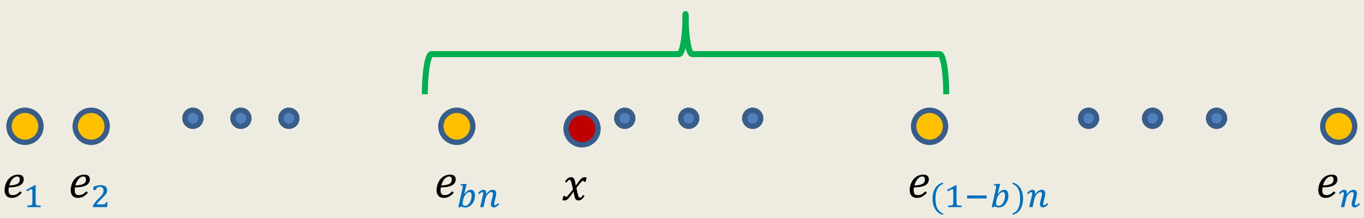
Solution for $T(n) = cn + T(an) + T(dn)$ is $O(n)$ **if** $\boxed{a + d < 1.}$

Concept of **approximate median**

Definition: Given a constant $0 < \boldsymbol{b} \leq \mathbf{1/2}$,

an element $\boldsymbol{x} \in \boldsymbol{S}$

if **rank(**$\boldsymbol{x}, \boldsymbol{S}$**)**



Learning from **QuickSelect**(S,i)

QuickSelect(S,i)

{ Pick an element x from S;

$(S_{<x}, S_{>x}, r) \leftarrow$ **Partition**(S,x);

O(n)

If(i = r) return x;

Else If (i< r)

QuickSelect($S_{<x}$,i)

Else

QuickSelect($S_{>x}$,$i - r$);

}

T($(1-b)n$)

Lesson 1

What is time complexity
of the algorithm

Answer: T(n) = cn + T($(1-b)n$)

= O(n)

Algorithm 2

Select($\boldsymbol{S}$,$\boldsymbol{i}$)

(A **linear time** algorithm)

Overview of the algorithm

Select(S,i)

{ Compute a b-approximate median, say x, of S; $\}$ **O**(n)

($S_{<x}$, $S_{>x}$, r)← **Partition**(S,x); $\}$ **O**(n)

If(i = r) **return** x;

Else If (i< r)

Select($S_{<x}$,i)

Else

Select($S_{>x}$,$i-r$);

}

$\}$ **T**($(1-b)n$)

Observation: If we can compute b-approximate median in **O**(n) time, we get **O**(n) time algo.

But that appears too much to expect from us. Isn't it ?

So what to do ☹ ?

Hint: use **Lesson 2**

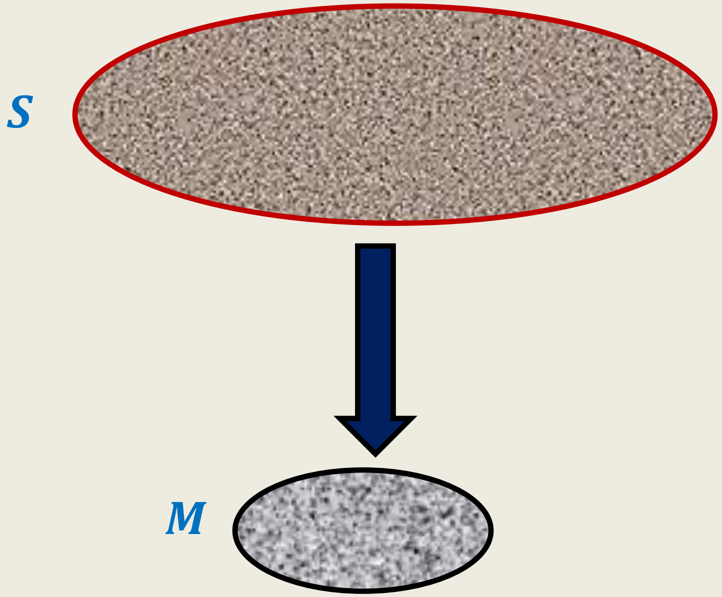
Observation: If we can compute b-approximate median

time complexity of the algorithm will still be **O**(n).

Spend some time on this observation to infer what it hints at.

AIM: How to compute a b-approximate median of S

in **O(**n)**+ T(**dn) time



Question: Can we form a set M of size dn such that

exact median of M is b-**approximate** median of S?

Forming the subset M with desired parameters

This step forms the core of the algorithm and is indeed a brilliant stroke of inspiration.
The student is strongly recommended to ponder over this idea from various angles.

- Divide S into **groups** of **5** elements;
- Compute median of each group by sorting;
- Let M be the set of medians;
- Compute median of M, let it be x;

O(n)

T($n/5$)

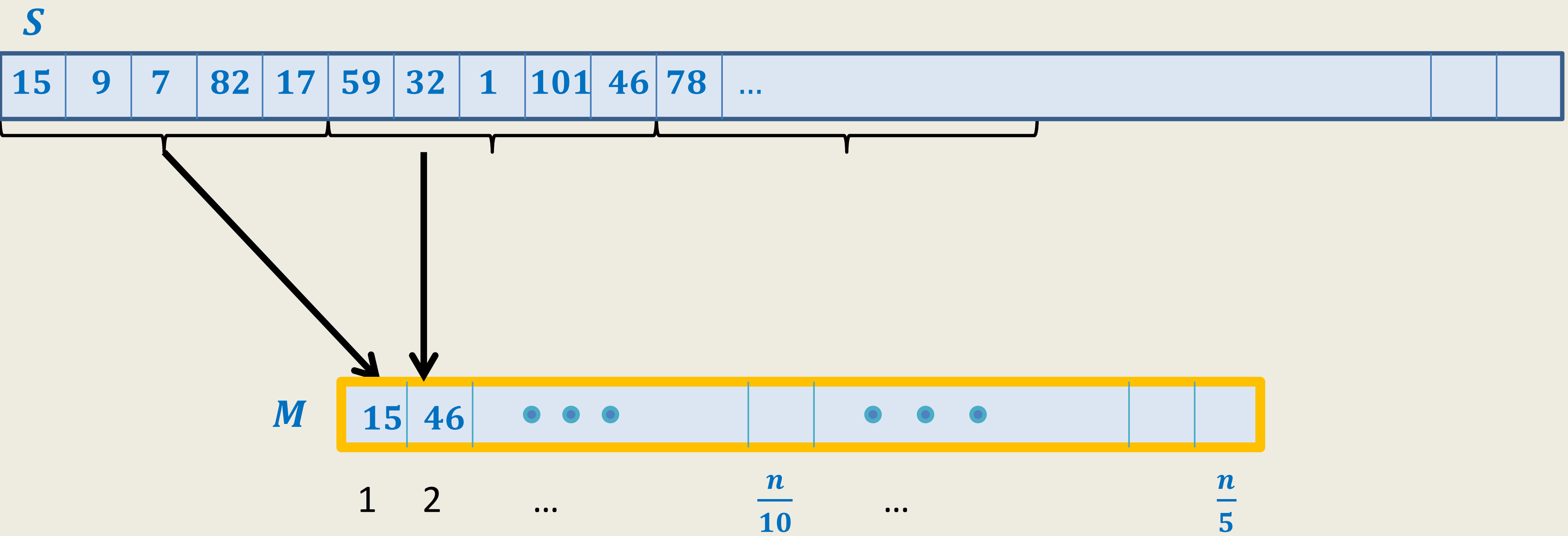
Question: Is x an approximate median of S ?

Answer: indeed.

The rank of x in M is $n/10$. Each element in M has two elements smaller than itself in its respective group. Hence there are at least $\frac{3n}{10} - 1$ elements in S which are **smaller** than x. In a similar way, there are at least $\frac{3n}{10} - 1$ elements in S which are **greater** than x. Hence, x is $\frac{3n}{10}$-approximate median of S.

(See the animation on the following slide to get a better understanding of this explanation.)

Forming the subset M



- Divide S into **groups** of **5** elements;
- Compute median of each group by sorting;

$\Big\}$ O(n)

Forming the subset M

S

15	9	7	82	17	59	32	1	101	46	78	…		

M

15	46	• • •	x	• • •			

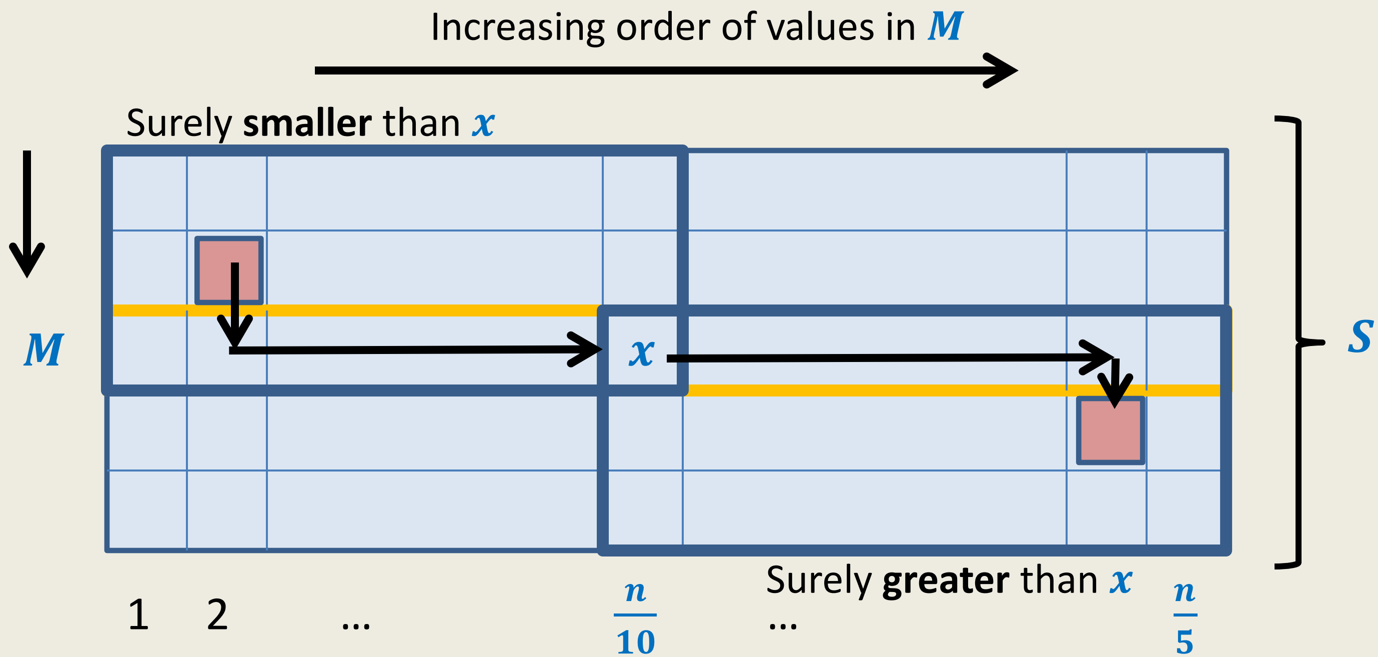
- Divide S into **groups** of **5** elements;
- Compute median of each group by sorting;
- Let M be the set of medians;
- Let x be median of M.

What can we say about rank of x in S ?

Spend some time to answer this question before moving ahead.

Forming the subset M

Bring back the remaining $\mathbf{4}$ elements associated with each element of M



➔ x is $\left(\frac{3n}{10}\right)$ −approximate median of S.

Time required to form M : $\mathbf{O}(n)$

Pseudocode for Select(S,i)

Select(S,i)
   $M \leftarrow \emptyset$;
   Divide S into **groups** of **5** elements;
   Sort each group and **add its median** to M; — O(n)
   $x \leftarrow$ Select(M,|M|/2); — T(n/5)
   ($S_{<x}$, $S_{>x}$, r) $\leftarrow$ Partition(S,x); — O(n)
   If(i = r) return x;
   Else If (i< r)
      Select($S_{<x}$,i)
    Else
      Select($S_{>x}$,$i - r$);

T(7n/10)

Analysis

T(**n**) = **cn + T**(**n/5**) + **T**(**7n/10**)

= **O**(**n**) **[Learning from Recurrence of type II]**

Theorem: Given any $\boldsymbol{S}$ of **n** elements, we can compute ith smallest element from $\boldsymbol{S}$ in O(**n**) worst case time.

Exercises

(Attempting these exercises will give you a better insight into the algorithm.)

- What is magical about number 5 in the algorithm ?
- What if we divide the set S into groups of size 3 ?
- What if we divide the set S into groups of size 7 ?
- What if we divide the set S into groups of even size (e.g. 4 or 6) ?