CIS 606 Analysis of Algorithms

Prune-and-Search

RATIONALE

- A prune-and-search algorithm is a method for finding an optimal value by iteratively dividing a search space into two parts: the promising one, which contains the optimal value and is recursively searched, and the second part without optimal value, which is pruned (thrown away). — Wikipedia

PRUNE-AND-SEARCH

- A Prune-and-Search algorithm contains three steps:
  - Divide: divide the input into two sets
  - Prune: based on certain criteria, one set is known not to include the solution and throw it away.
  - Conquer:
    - Solve the problem on the remaining set recursively
  - Base case: if the size of the set is small enough, solve it in a straightforward way.

BINARY SEARCH

- Input:
  - an array A[1… n] of values in the ascending order and a value x
- Output: the index of x in A
- 1. Divide:
  - Divide A[1… n] into two subarrays A[1… n/2] and A[n/2+1, …, n]
- 2. Prune:
  - If A[n/2] >=x, prune A[n/2+1, …, n]
  - If A[n/2] <x, prune A[1… n/2]
- 3. Conquer:
  - Continue searching x in the remaining subset;   (binarysearch(x, A[1…n/2]))
  - Base case: if A is has only one element left, return true if it equals x; otherwise, return false.

BINARY SEARCH (CONT)

Operation: Search 34 in the given sorted array A

	L				m				R
	5	7	9	33	65	67	345	378	4777
Index	0	1	2	3	4	5	6	7	8

Base case:
33<34

BINARY SEARCH

Goal: Binary search x in the sorted array A

Output: the index of x in A

```
BiSearch (A, L, R, x)
{
    if L = R                                O(1)
        if A[L] = x     return L            O(1)
        else            return -1           O(1)
    m = (L+R)/2                             O(1)
    if A[m] >= x                            O(1)
        R = m                               O(1)
        return BiSearch (A, L, R, x)        T(n/2)
    else
        L = m                               O(1)
        return BiSearch (A, L, R, x)        T(n/2)
}
```

Call BiSearch(A, 1, n, x) to start the algorithm

Recurrence:

$$T(n) = T(\frac{n}{2}) + O(1) \quad \text{for } n > 1$$

$$T(n) = O(1) \quad \text{for } n = 1$$

What does T(n) equals finally?

SELECTION PROBLEM

```
Select(A, k)
{
  mergesort(A) O(nlogn)
  Return A[k] O(1)
}
O(n log n)
```

- **Input:**
  - **An array A[1… n] and an integer k with 1<= k<= n**
- **Output:**
  - **The k-th smallest element x* in A**
- **Assumption: no two numbers are equal**
- **For example:**
- **You have k-1 number in A less than the k-th smallest number.**
  - **A: 8 25 3 37 12 16 7 22**
  - **Given k=4 and x=12**
- **Note when k= $\lfloor n/2 \rfloor$, the k-th smallest number of A is its median.**

SOLUTIONS

- **Input:**
  - **An array A[1… n] and an integer k with 1<=k<=n**
- **Output:**
  - **The k-th smallest element x* of A**

```
kSmall(A, k)
{
   Apply MergeSort to A; O(nlog n)
   Return A[k]; O(1)
}
O(n logn)
```

```
kLinearScan(A, k)
{
   S=+∞
   m=-∞
   for i = 1 to k
       for j =1 to n
           if m<A[j]<S
               S=A[j]
       m = S
       S = +∞
}
```

ALGORITHMS

- Input:
  - an array A[1… n] and an integer k with 1<=k<=n
- Output:
  - the k-th smallest element x* of A
- Algorithms:
  - Sorting then choose A[k] — O(n log n)
  - k linear scans — O(kn)
  - Selection Algorithms — O(n)
  - Min-heap: — O(n+klog n)

SELECTION ALGORITHM — DIVIDE

6	-1	9	5	4	3	8	-5	-9	7

- Divide:
  - How to partition A[1, …, n] into two sets?
    - A pivot is needed. E.g., in binary search, the pivot is the middle position.
    - The standard of partition.

SELECTION ALGORITHM — DIVIDE (CO

```
partition(A)
{
   P=A[1]
   For i=2:n
     If A[i]>P
         Put A[i] into array
A2
     Else
```


- **Divide:**
  - **Pivot: Pick the first element as the pivot x**
  - **Partition Standard: Partition A[1, …, n] into A_1, pivot x, and A_2 where A_1 contains all< x and A_2 contains all >x**

```
Selection(A, 9)
{
    Partition in A1 and A2
    m=6 m+1=7
    K=9>7
```

SELECTION ALGORITHM — PRUNE

Pivot
x

-1	5	4	3	-5	-9		6		9	8	7

A_1 A_2

- **Prune:**
  - **Based on certain criteria(observations), we determine which of A_1 and A_2 contains x*?**
  - **We have 6 numbers in A1, all numbers in A1 are smaller than pivot. We conclude the rank of pivot in A is 7, the pivot is the 7th smallest number in A.**

Let m be the size of A1. In A, we totally have m numbers <pivot and those m numbers are all in A1. All other numbers of A larger than the pivot are in A2.

The pivot is the (m+1)-th smallest number. K-th smallest number.

If k = (m+1), The k-th smallest number is this pivot. Return pivot.
A[1] A[n/2]

If k < (m+1), the k-th smallest number is smaller than the (m+1) -th smallest number, pivot, we prune A2. Continue search the k-th smallest number of A in A1. Because all numbers of A less than the k-th smallest number are in A1, the k-th smallest number of A is the k-th smallest number of A1. Find the k-th smallest number of A1. We do

Selection(A1, k).

If k > (m+1), the k-th smallest number is larger than the (m+1)-th smaller number, we are allowed to throw away A1. By this, we prune (m+1) numbers of A less than the k-th smallest number. Among all left numbers, all of which are in A2, there are k-(m+1) number smaller than the k-th smallest number of A. Thus, the k-th smallest number of A is the (k-(m+1))th smallest number of A2. Continue our search in A2 by

K-th smallest number of A —-
in A there are k-1 numbers < the k-th smallest number of A.
A: A1, pivot, and A2.   The pivot is (m+1)-th smallest number of A.
k>m+1 —- the k-th smallest number of A is larger than the pivot.
A1 contains all numbers of A less than pivot.
All number of A1 & pivot < the k-th smallest number of A.
If we throw away A1 and the pivot from A, we prune m+1 numbers of A smaller than the k-th smaller number of A and A2 contains all left elements. Among all left numbers, there are totally (k-1)-(m+1) of numbers less than the k-th smallest number of A. In other words, in A2, there are totally (k-1)-(m+1) numbers left which are smaller the k-th smallest number of A. The rank of the k-th smallest number of A in A2 become (k-1)-(m+1)+1=k-m-1 where m is the size of A1.

SELECTION ALGORITHM — PRUNE (CONT)

Pivot
x

-1	5	4	3	-5	-9	6	9	8	7

A_1 A_2

- **Prune:**
  - **Observation:**
    - **If k = $|A_1|$+1, pivot x is x***
    - **If k < $|A_1|$+1, x* is in A_1 and still the k-th smallest number in A_1**
    - **If k > $|A_1|$+1, x* is in A_2 but the (k-$|A_1|$-1)-th smallest number in A_2**
      E.g., k = 9

SELECTION ALGORITHM — CONQUER

Pivot
x

6	-1	5	4	3	-5	-9

A_1

9	8	7

A_2

- **Conquer:**
  - **If $k < |A_1|+1$, continue search in A_1 by a recursive call Selection (A_1, k).**
  - **If $k > |A_1|+1$, continue search in A_2 by a recursive call Selection (A_2, $k-|A_1|-1$).**
  - **Base case:**
    - **If $|A|<=5$, find the k-th smallest number directly in a straightforward way.**
    - **Sorting A first and return A[k];**

SELECTION ALGORITHM

Pivot
x

6	-1	9	5	4	3	8	-5	-9	7

```
Selection(A, k)    //return k-th smallest number in A
{
      if |A|<=5  {sort A; return A[k];}
      Pick a pivot x in A
      Partition A into A1 and A2 where A1 contains all elements of A < x and
                                               A2 contains all elements in A > x.
      if k = |A1|+1      return x
      else if k< |A1|+1     return Selection (A1, k)
      else   return Selection (A2, k-|A1|-1)
```

SELECTION ALGORITHM — RUNNING TIME

```
Selection(A, k)                                          // T(n)
{
    if |A|<=5  {sort A; return A[k];}                    // O(1)
    Pick a pivot x in A;                                 // O(1)
    Partition A into A1 and A2 s.t A1 contains all elements of A < x and
                                  A2 contains all elements in A > x.
                                                         // O(n)
    if k = |A1|+1     return x;                          // O(1)
    else if k< |A1|+1
            return Selection (A1, k)                     // T(|A1|) = T(m)
    else
            return Selection (A2, k-|A1|-1)              // T(|A2|) = T(n-m-1)
}
```

$$T(n) = \max\{T(|A_1|), T(|A_2|)\} + O(n) = T(\max\{|A_1|, |A_2|\}) + O(n) <= T(\frac{9}{10}n) + O(n)$$

WORST CASE — IF ALWAYS PICKING THE SMALLEST NUMBER AS PIVOT

Pivot x

$k=10$

$|A|=10$

0	1	2	3	4	5	6	7	8	9

Pivot x

0

$|A_1|=0$

1	2	3	4	5	6	7	8	9

A_2

$|A_2|=9$

$$T(n) = T(\max\{0, n-1\}) + O(n) = T(n-1) + O(n)$$

WORST CASE RUNNING TIME

$$
\begin{aligned}
T(n) &= T(n-1) + n \\
&= T(n-2) + (n-1) + n \\
&= T(n-3) + (n-2) + (n-1) + n \\
&= \cdots \\
&= T(1) + 2 + 3 + \cdots + (n-3) + (n-2) + (n-1) + n \\
&= 1 + 2 + 3 + \cdots + (n-3) + (n-2) + (n-1) + n \\
&= \sum_{i=1}^{n} i = n(n+1)/2 \\
&= O(n^2)
\end{aligned}
$$

IF ALWAYS LUCKILY GETTING MEDIAN AS PIVOT

— $|A_1| = |A_2| = \frac{n}{2}$

$$
\begin{aligned}
T(n) &= \max\{T(|A_1|), T(|A_2|)\} + O(n) \\
&= T(\frac{n}{2}) + n \\
&= T(\frac{n}{2^2}) + \frac{n}{2} + n \\
&= T(\frac{n}{2^3}) + \frac{n}{2^2} + \frac{n}{2} + n \\
&= \cdots \\
&= T(\frac{n}{2^k}) + \frac{n}{2^{(k-1)}} + \frac{n}{2^2} + \frac{n}{2} + n \\
&= n \cdot (\frac{1}{2^k} + \frac{1}{2^{(k-1)}} + \frac{1}{2^2} + \frac{1}{2^1} + 1) \\
&= O(n)
\end{aligned}
$$

If we always luckily get the median of A as pivot

Partition A into two equal subsets every iteration

But we are not always lucky to meet the median.

Computing the median takes O(n logn) without this selection algorithm.

IF A GOOD PIVOT IS SELECTED S.T. $\max\{T(|A_1|), T(|A_2|)\} \le \frac{n}{b}$, e.g., $b = \frac{10}{9}$

$$
\begin{aligned}
T(n) &= \max\{T(|A_1|), T(|A_2|)\} + O(n) \\
&= T(\frac{9n}{10}) + n \\
&= T((\frac{9}{10})^2 \cdot n) + \frac{9}{10} \cdot n + n \\
&= \cdots \\
&= T((\frac{9}{10})^k \cdot n) + (\frac{9}{10})^k \cdot n + \cdots + \frac{9}{10} \cdot n + n \\
&= T(1) + (\frac{9}{10})^{k-1} \cdot n + \cdots + \frac{9}{10} \cdot n + n \\
&= 1 + n \cdot [(\frac{9}{10})^{k-1} + \cdots + \frac{9}{10} + 1] \\
&= O(n)
\end{aligned}
$$

Pick a specific value in A as pivot such that x* is in the subset whose size is no more than $\frac{9n}{10}$

COMPUTING A GOOD PIVOT — PARTITION

36	42	18	1	5	13	25	27	14	29	23	12	19	33	8	6	28	50	16	21	32	15	26	43	3

36	13	23	6	32
42	25	12	28	15
18	27	19	50	26
1	14	33	16	43
5	29	8	21	3

Partition A into groups of 5 elements

$\frac{n}{5}$ groups

$O(n)$

COMPUTING A GOOD PIVOT — SORTING EACH GROUP

36	42	18	1	5	13	25	27	14	29	23	12	19	33	8	6	28	50	16	21	32	15	26	43	3

1	13	8	6	3
5	14	12	16	15
18	25	19	21	26
36	27	23	28	32
42	29	33	50	43

Time: $O(1) * \frac{n}{5} = O(n)$

COMPUTING A GOOD PIVOT — COMPUTING MEDIANS

36	42	18	1	5	13	25	27	14	29	23	12	19	33	8	6	28	50	16	21	32	15	26	43	3

	1	13	8	6	3
	5	14	12	16	15
M	18	25	19	21	26
	36	27	23	28	32
	42	29	33	50	43

Pivot

Compute the median of each group

Time: $O(1) * \frac{n}{5} = O(n)$

Compute the median of all medians by calling Selection(M, |M|/2)

Time: $T(\frac{n}{5})$

THE MEDIAN OF MEDIANS IS A GOOD PIVOT

```
CGPivot(A)
{
    Divide A into n/5 groups with each of size O(5)          //O(n)
    Sort each group               //Sorting each group takes O(1) and so O(n) in total.
    Compute the median of each group                          //O(n)
    Create an array M to store all medians                    //O(n)
    return Selection(M, M/2)  // Compute the median of M      — T(n/5)
}
```

Time: $T(\frac{n}{5})$+O(n)

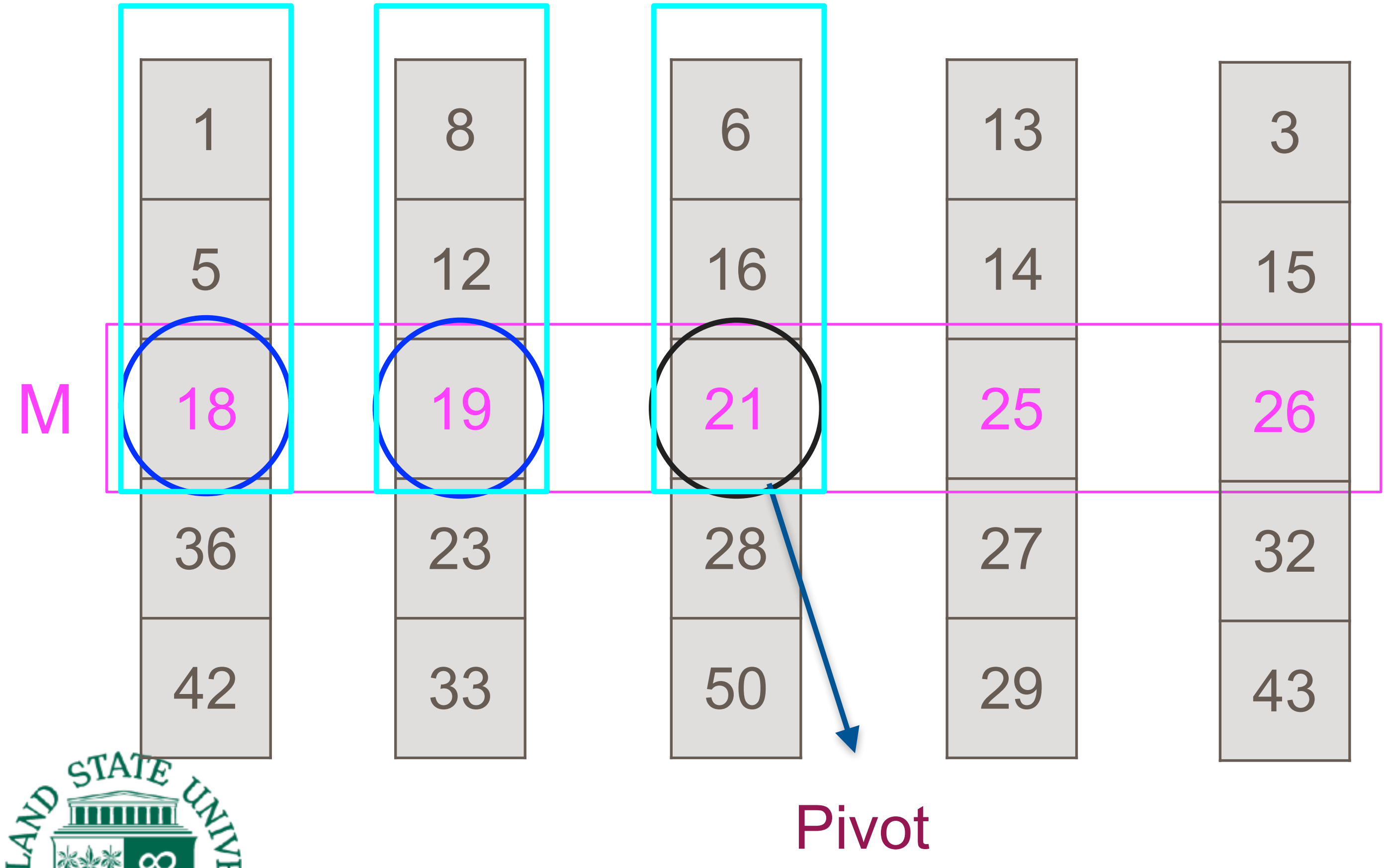
SELECTION ALGORITHM

```
Selection(A, k)                                         // T(n)
{
    if |A|<=5       {sort A; return A[k];}              // O(1)
    x = CGPivot(A)                                      // T(n/5) + O(n)
    Partition A by x into A1 and A2 s.t A1 contains all elements of A < x and
                             A2 contains all elements in A > x.          // O(n)
    if k = |A1|+1      return x                         // O(1)
    else if k< |A1|+1
          return Selection (A1, k)                      // T(|A1|)
    else
          return Selection (A2, k-|A1|-1)               // T(|A2|)
}
```

$$T(n) = T(\max\{|A_1|, |A_2|\}) + T(\frac{n}{5}) + O(n)$$

$$T(n) <= T(\frac{7}{10} \cdot n) + T(\frac{n}{5}) + O(n)$$

A GOOD PIVOT IS THE MEDIAN OF MEDIANS: $|A_1| \geq \frac{3n}{10}$



Medians of half groups are smaller than Pivot

— $\frac{n}{5} * \frac{1}{2}$ groups

In each such group, at least 3 elements are smaller than Pivot.

—- at least $\frac{n}{5} * \frac{1}{2} * 3$ elements

Partition A into A_1 and A_2 by Pivot s.t. these smaller elements are in A_1:

$|A_1| >= \frac{n}{5} * \frac{1}{2} * 3$ elements

A GOOD PIVOT IS THE MEDIAN OF MEDIANS:

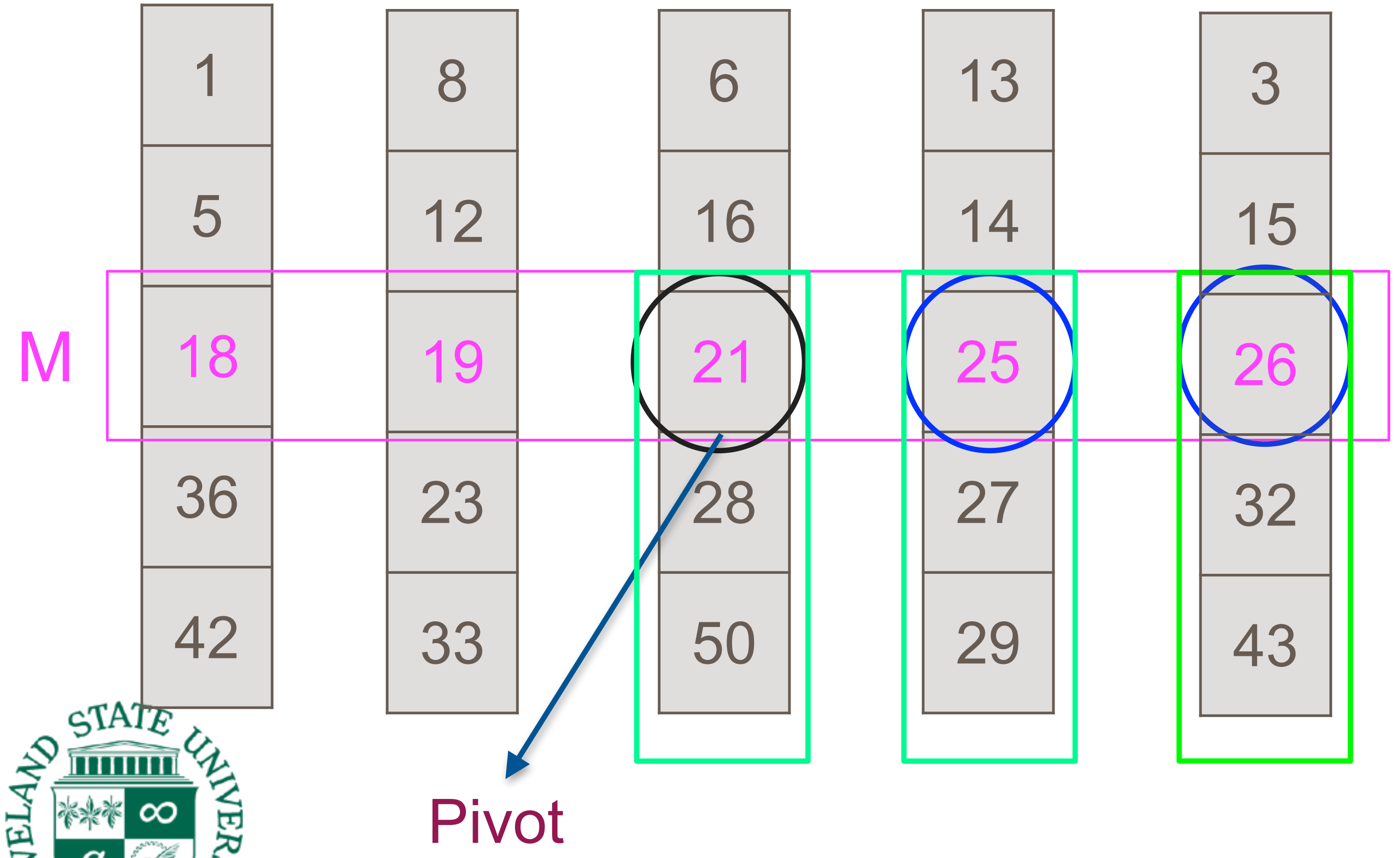
$|A_2|$ HAS AN UPPER BOUND $\frac{7n}{10}$

Partition A into A_1 and A_2 by it s.t. these smaller elements are in A_1:

$|A_1| >= \frac{n}{5} * \frac{1}{2} * 3$ elements

→ $|A_1| \geq \frac{3n}{10}$ → $n - |A_2| - 1 \geq \frac{3n}{10}$ → $|A_2| \leq \frac{7n}{10}$

A GOOD PIVOT IS THE MEDIAN OF MEDIANS: $|A_2| \geq \frac{3n}{10}$



Medians of half groups are larger than Pivot

— $\frac{n}{5} * \frac{1}{2}$ groups

In such each group, at least 3 elements are larger than Pivot.

—- at least $\frac{n}{5} * \frac{1}{2} * 3$ elements

Partition A into A_1 and A_2 by it s.t. these larger elements are in A_2:

$|A_2| >= \frac{n}{5} * \frac{1}{2} * 3$ elements

A GOOD PIVOT IS THE MEDIAN OF MEDIANS:

$|A_1|$ HAS AN UPPER BOUND $\frac{7n}{10}$

Partition A into A_1 and A_2 by it s.t. these larger elements are in A_2:
$|A_2| >= \frac{n}{5} * \frac{1}{2} * 3$ elements

→ $|A_2| \geq \frac{3n}{10}$ → $n - |A_1| - 1 \geq \frac{3n}{10}$ → $|A_1| \leq \frac{7n}{10}$

A GOOD PIVOT IS THE MEDIAN OF MEDIANS:

BOTH $|A_1|$ AND $|A_2|$ HAVE AN UPPER BOUND $\frac{7n}{10}$

Partition A into A_1 and A_2 by it s.t. these smaller elements are in A_1:

$|A_1| >= \frac{n}{5} * \frac{1}{2} * 3$ elements

$\rightarrow |A_1| \geq \frac{3n}{10} \rightarrow n - |A_2| \geq \frac{3n}{10} \rightarrow |A_2| \leq \frac{7n}{10}$

Partition A into A_1 and A_2 by it s.t. these larger elements are in A_2:

$|A_2| >= \frac{n}{5} * \frac{1}{2} * 3$ elements

$\rightarrow |A_2| \geq \frac{3n}{10} \rightarrow n - |A_1| \geq \frac{3n}{10} \rightarrow |A_1| \leq \frac{7n}{10}$

$$\frac{3n}{10} \leq |A_1| \leq \frac{7n}{10}$$

$$\frac{3n}{10} \leq |A_2| \leq \frac{7n}{10}$$

$$\max(|A_1|, |A_2|) \leq \frac{7}{10} \cdot n$$

SELECTION ALGORITHM

```
Selection(A, k)                                         // T(n)
{
    if |A|<=5      {sort A; return A[k];}               // O(1)
    CGPivot(A)                                          // T(n/5) + O(n)
    Partition A into A1 and A2 s.t A1 contains all elements of A < x and
                           A2 contains all elements in A > x.        // O(n)
    if k = |A1|+1      return x                         // O(1)
    else if  k< |A1|+1
           return Selection (A1, k)                     // T(|A1|)
    else
           return Selection (A2, k-|A1|-1)              // T(|A2|)
}
```

$$T(n) = T(\max\{\,|A_1|,|A_2|\,\}) + T(\frac{n}{5}) + O(n) \le T(\frac{7}{10}\cdot n) + T(\frac{n}{5}) + O(n)$$

SOLVING THE RECURRENCE

$$T(n) = T(\frac{7}{10} \cdot n) + T(\frac{n}{5}) + O(n)$$

Guess $T(n) = O(n)$

1. Assume it is true for $T(\frac{7}{10} \cdot n)$ and $T(\frac{n}{5})$, i.e., $T(\frac{7}{10} \cdot n) \leq c \cdot \frac{7n}{10}$ and $T(\frac{n}{5}) \leq c \cdot \frac{n}{5}$ for a constant $c > 0$ and $n > n_0$

2. Verification:

$$T(n) = T(\frac{7}{10} \cdot n) + T(\frac{n}{5}) + n \leq c \cdot \frac{7n}{10} + c \cdot \frac{n}{5} + n = c \cdot \frac{9n}{10} + n$$

$$= c \cdot n - c \cdot \frac{n}{10} + n = cn - (\frac{c}{10} - 1)n$$

$$T(n) \leq cn - (\frac{c}{10} - 1) \cdot n$$

$$(\frac{c}{10} - 1) > 0 \qquad T(n) \leq cn \qquad T(n) = O(n)$$

$$c > 10 \text{ and } \ n_0 = 1$$