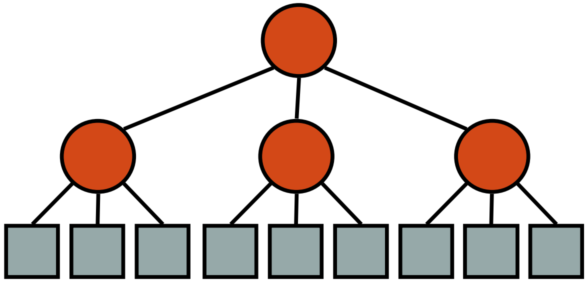
Analysis of Algorithm
Lecture-04: Merge Sort and Selection Problem

Sorting

- Sorting is a well studied problem from analysis point of view
- Sorting is one of few problems where provable lower bounds exist on how fast we can sort
- In sorting we are given an array A[1..n] of n numbers
- We are to reorder these elements into increasing (or decreasing) order

Divide-and-Conquer

- The Divide-and-Conquer strategy is employed to solve large number of computational problems.
  - Divide: the problem into a small number to pieces.
  - Conquer: solve each piece by applying divide and conquer to it recursively.
  - Combine: the pieces together into a global solution.



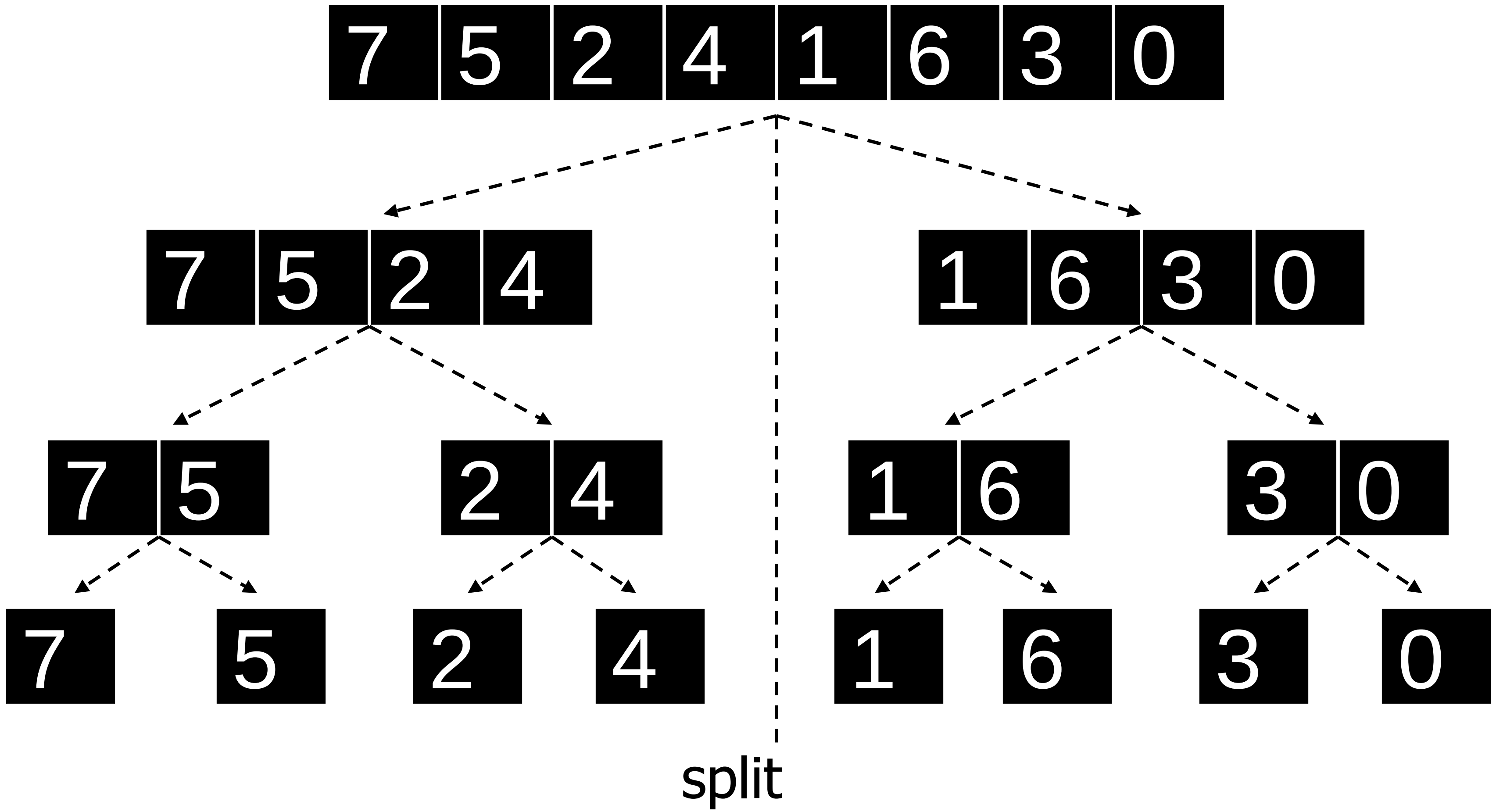
Merge Sort

- Divide: Split A down the middle into two subsequences, each of size roughly n/2
- Conquer: Sort each subsequence by calling merge sort recursively on each
- Combine: Merge the two sorted subsequences into a single sorted list.

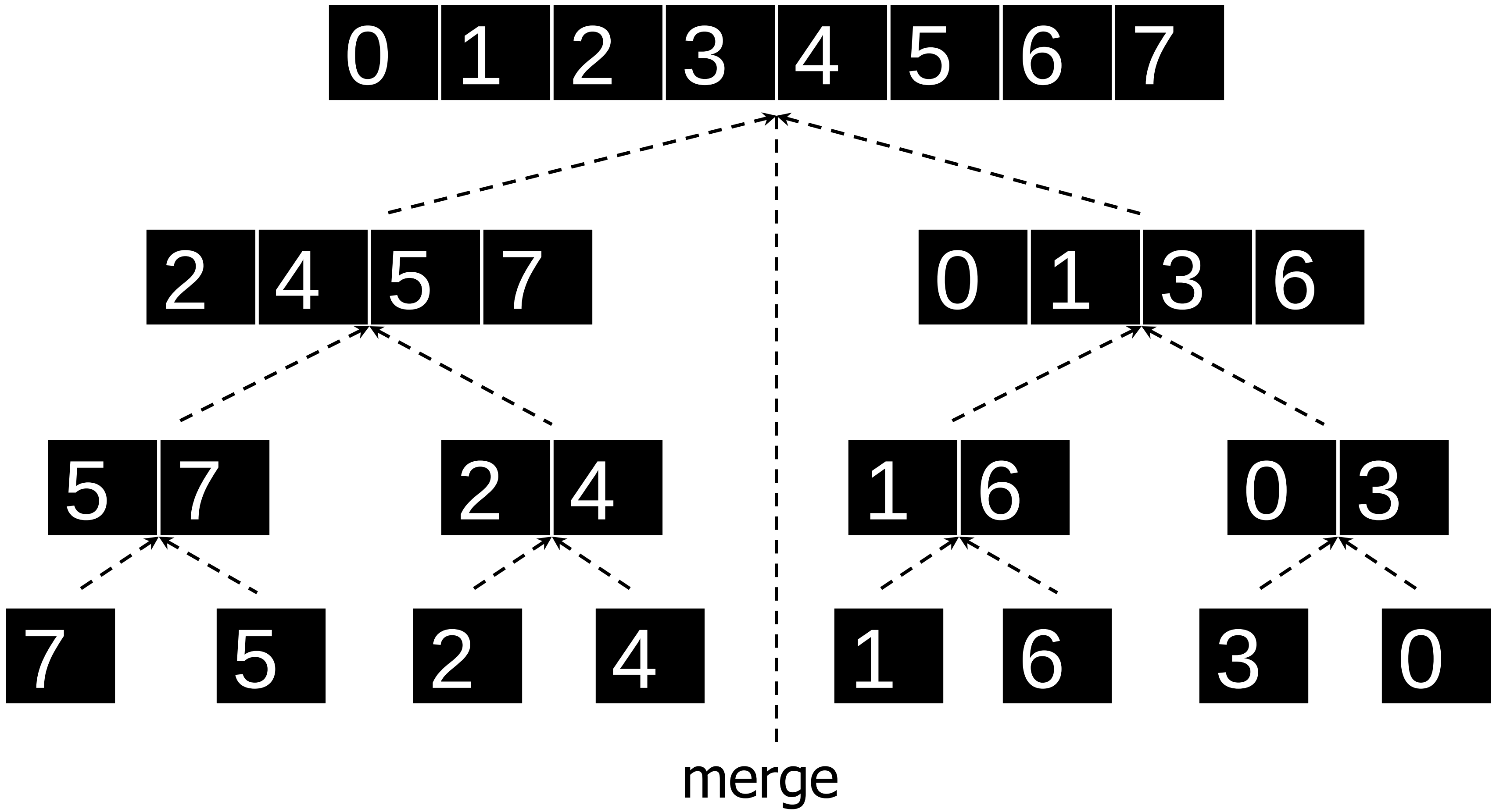
Merge Sort

- The dividing process ends when we have split the subsequences down to a single item.
- A sequence of length 1 is trivially sorted.

Merge Sort (split)



Merge Sort (merge)



MERGE-SORT Algorithm

```
MERGE-SORT(array A, int p, int r)
if (p < r) then
  q ←(p+r)/2
  MERGE-SORT(A, p, q)
  MERGE-SORT(A, q+1, r)
  MERGE(A, p, q, r)
```

MERGE Algorithm

```
MERGE(array A, int p, int q, int r)
  int B[p..r]; int i ← k  ← p; int j ← q+1;
  while (i ≤ q) and (j ≤ r)
  do
        if (A[i] < A[j]) then
                B[k++] ← A[i++];
        else
                B[k++] ← A[j++];
  while (i ≤ q) do B[k++] ← A[i++];
  while (j ≤ r) do B[k++] ← A[j++];
  for i ← p to r do  A[i] ← B[i];
```

Logarithmic identities

$$y = \log_b(x) \text{ if and only if } x = b^y$$

$$\log_b(xy) = \log_b(x) + \log_b(y)$$

$$\log_b\left(\frac{x}{y}\right) = \log_b(x) - \log_b(y)$$

$$\log_b(x^y) = y \log_b(x)$$

$$\log_b\left(\sqrt[y]{x}\right) = \frac{\log_b(x)}{y}$$

$$x^{\log(y)} = y^{\log(x)}$$

$$\log_b(b) = 1$$

$$\log_b(1) = 0$$

Analysis of Merge Sort

- First consider the running time of procedure MERGE(A, p, q, r).
- Let $n = r - p + 1$ denote the total length of both left and right sub arrays, i.e. sorted pieces.
- The MERGE procedure contains four loops, none nested in the other.
- Each loop can execute at most n times.
- Let $T(n)$ denote the worst case running time of MERGE-SORT on an array of length n.
- If we call MERGE-SORT with an array containing a single item ($n = 1$) then the running time is constant.
- We can just write $T(n) = 1$, ignoring all constants.

Analysis of Merge Sort (cont'd)

- For n > 1 MERGE-SORT Splits into two halves, sorts the two and then merges them together.
- The left half is of size $\lceil n/2 \rceil$ and the right half is $\lfloor n/2 \rfloor$
- How long does it take to sort the elements in sub array of size
- We do not know this, $\lceil n/2 \rceil$ but because,

$$\lceil n/2 \rceil < n \quad \text{for} \quad n > 1$$

we can express this as

$$T(\lceil n/2 \rceil)$$

Analysis of Merge Sort (cont'd)

- Similarly the time taken to sort right sub array is expressed as $T(\lfloor n/2 \rfloor)$
- In conclusion we have

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n & \text{otherwise} \end{cases}$$

- This is called recurrence relation i.e. a recursively defined function.

Solving the Recurrence

- Lets expand the terms

$$T(1) = 1$$
$$T(2) = T(1) + T(1) + 2 = 1 + 1 + 2 = 4$$
$$T(3) = T(2) + T(1) + 3 = 4 + 1 + 3 = 8$$
$$T(4) = T(2) + T(2) + 4 = 8 + 8 + 4 = 12$$
$$T(5) = T(3) + T(2) + 5 = 8 + 4 + 5 = 17$$
$$\ldots$$
$$T(8) = T(4) + T(4) + 8 = 12 + 12 + 8 = 32$$
$$\ldots$$
$$T(16) = T(8) + T(8) + 16 = 32 + 32 + 16 = 80$$
$$\ldots$$
$$T(32) = T(16) + T(16) + 32 = 80 + 80 + 32 = 192$$

Solving the Recurrence (cont'd)

- What is the pattern here?
- Let's consider the ratio of T(n)/n for powers of 2

$$T(1)/1 = 1$$
$$T(2)/2 = 2$$
$$T(4)/4 = 3$$
$$T(8)/8 = 4$$
$$T(16)/16 = 5$$
$$T(32)/32 = 6$$

- This suggests $T(n)/n = \log n + 1$

Solving the Recurrence (cont'd)

- Floor and ceiling are difficult to deal with
- If n is assumed to be a power of 2 ($2^k = n$) this will simplify the recurrence to

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(n/2) + n & \text{otherwise} \end{cases}$$

The Iteration Method

$$T(n) = 2T(n/2) + n$$

$$= 2(2T(n/4) + n/2) + n$$

$$= 4T(n/4) + n + n$$

$$= 4(2T(n/8) + n/4) + n + n$$

$$= 8T(n/8) + n + n + n$$

$$= 8(2T(n/16) + n/8) + n + n + n$$

$$= 16T(n/16) + n + n + n + n$$

The Iteration Method (cont'd)

- If n is power of 2 then let n = 2^k or k=log n.

$$T(n) = 2^k T(n/(2^k)) + \underbrace{(n + n + n + \ldots + n)}_{\text{k times}}$$

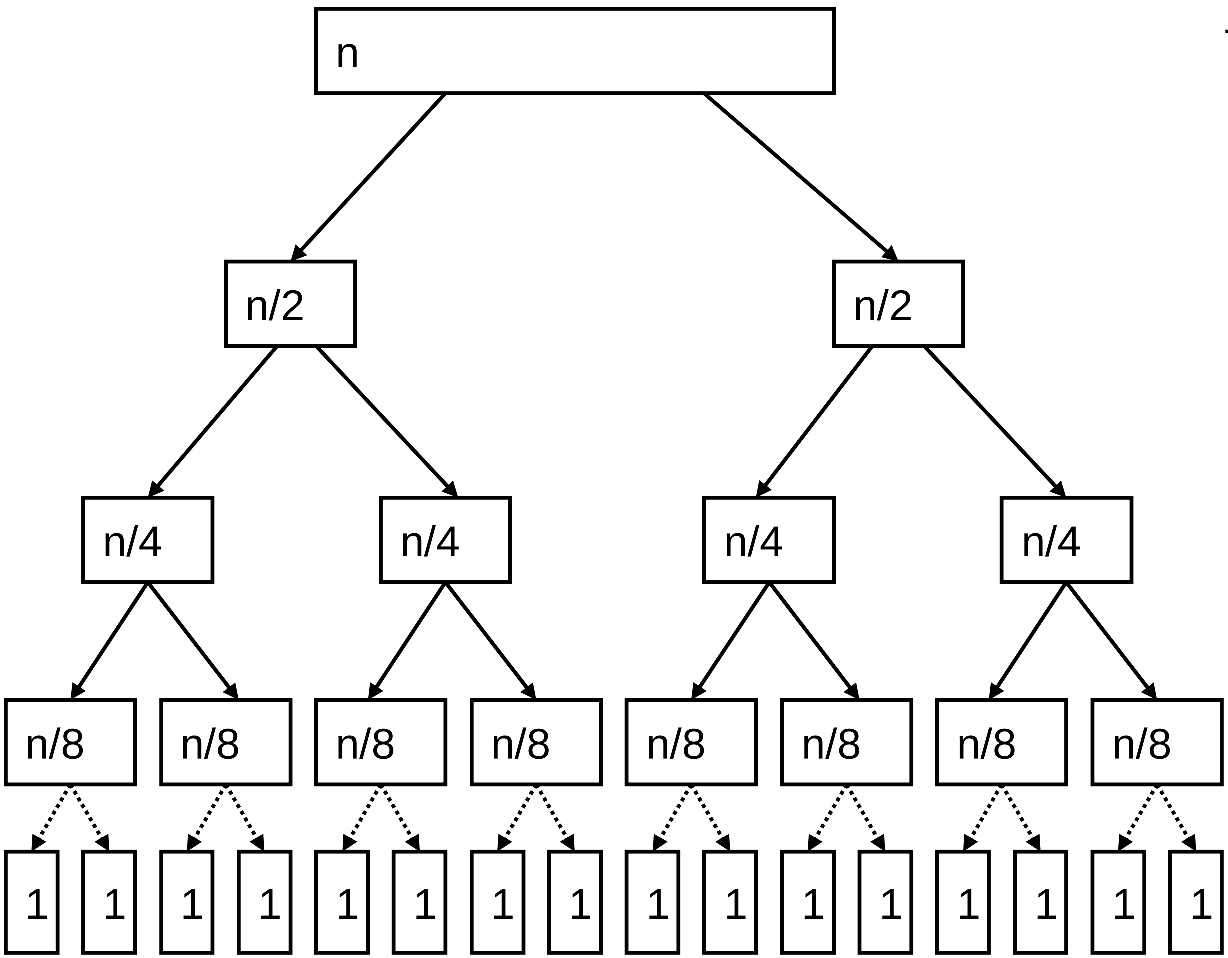
$$= 2^k T(n/(2^k)) + kn$$

$$= 2^{(\log n)} T(n/(2^{(\log n)})) + (\log n)n$$

$$= 2^{(\log n)} T(n/n) + (\log n)n$$

$$= nT(1) + n\log n = n + n\log n$$

MERGE-SORT Recursion tree



Time to merge

$$= n + 2(n/2)$$

$$= n + 4(n/4)$$

$$= n + 8(n/8)$$

$$= n + n(n/n)$$

$$= n$$

$$= n(\log(n) + 1)$$

Log(n)+1 Levels

Example-1: Solving Recurrence

- Assume n to be power of 4 i.e. $n = 4^k$ and $k = \log_4 n$

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 3T(n/4) + n & \text{otherwise} \end{cases}$$

Example-1: Solving Recurrence (cont'd)

- Iteration Method

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

$$= 3(3T(n/16)) + (n/4)) + n$$

$$= 9T(n/16) + 3(n/4) + n$$

$$= 27T(n/64) + 9(n/16) + 3(n/4) + n$$

$$= \ldots$$

Example-1: Solving Recurrence (cont'd)

$$T(n) = 3^k T(\frac{n}{4^k}) + 3^{k-1}(n/4^{k-1}) + \ldots + 9(n/16) + 3(n/4) + n$$

$$T(n) = 3^k T(\frac{n}{4^k}) + \sum_{i=0}^{k-1} \frac{3^i}{4^i} n$$

With $n = 4^k$ and $T(1) = 1$

$$T(n) = 3^{\log_4 n} T(1) + \sum_{i=0}^{(\log_4 n)-1} \frac{3^i}{4^i} n$$

Example-1: Solving Recurrence (cont'd)

n remains constant

throughout the sum and $3^i / 4^i = (3/4)^i$;

we thus have

$$T(n) = n^{\log_4 3} + n \sum_{i=0}^{(\log_4 n)-1} \left(\frac{3}{4}\right)^i$$

we used the formula $a^{\log_b n} = n^{\log_b a}$

The sum $\sum_{i=0}^{(\log_4 n)-1} \left(\frac{3}{4}\right)^i$ is a geometric series;

for $x \neq 1$

$$\sum_{i=0}^{m} x_i = \frac{x^{m+1} - 1}{x - 1}$$

Example-1: Solving Recurrence (cont'd)

$$\sum_{i=0}^{m} x_i = \frac{x^{m+1}-1}{x-1}$$

In this case $x = 3/4$ and $m = \log_4 n - 1$

We get

$$T(n) = n^{\log_4 3} + n\frac{(3/4)^{(\log_4 n)-1+1}-1}{(3/4)-1}$$

Applying the log identity once more

$$(3/4)^{\log_4 n} = n^{\log_4 (3/4)} = n^{\log_4 3 - \log_4 4}$$

$$= n^{\log_4 3 - 1} = \frac{n^{\log_4 3}}{n}$$

Example-1: Solving Recurrence (cont'd)

if we plug this back, we get

$$T(n) = n^{\log_4 3} + n\frac{\frac{n^{\log_4 3}}{n} - 1}{(3/4) - 1}$$

$$= n^{\log_4 3} + \frac{n^{\log_4 3} - n}{-1/4}$$

$$= n^{\log_4 3} + \frac{n^{\log_4 3} - n}{-1/4} = n^{\log_4 3} + \frac{(n^{\log_4 3} - n)/1}{-1/4} = n^{\log_4 3} + \frac{4(n^{\log_4 3} - n)}{-1} = n^{\log_4 3} + 4(-n^{\log_4 3} + n)$$

$$= 4n - 4n^{\log_4 3} + n^{\log_4 3}$$

$$= 4n - 3n^{\log_4 3}$$

with $\log_4 3 \approx 0.79$, we finally have the result!

$$T(n) = 4n - 3n\log_4 3 \approx 4n - 3n^{0.79} \in \Theta(n)$$

Example-2: Solving Recurrence

$$f(n) = f(n\text{-}1) + n$$

$$= f(n\text{-}2) + n - 1 + n$$

$$= f(n\text{-}3) + n - 2 + n - 1 + n$$

$$= f(n\text{-}4) + n - 3 + n - 2 + n - 1 + n$$

$$\ldots$$

$$= f(1) + 2 + 3 + 4 + \ldots + n - 1 + n$$

$$= \frac{n(n+1)}{2} = \Theta(n^2)$$

Example-3: Solving Recurrence

$$f(n) = f(n/2) + 1$$

$$= f(n/4) + 1 + 1$$

$$= f(n/8) + 1 + 1 + 1$$

$$= f(n/16) + 1 + 1 + 1 + 1$$

$$\ldots$$

$$= f(n/n) + 1 + 1 + \ldots + 1$$

$$= \log_2 n$$

$$= \Theta(\log n)$$

Example-4: Solving Recurrence

$$f(n) = 2f(n/2) + 1$$
$$= 2(2f(n/4)+1)+1$$
$$= 4f(n/4)+2+1$$
$$= 4(2f(n/8)+1)+2+1$$
$$= 8f(n/8)+4+2+1$$
$$= 8(2f(n/2)+1)+4+2+1$$
$$= 16f(n/16) + 8 + 4 + 2 + 1$$

...

$$= nf(n/n)+n/2+\ldots+8+4+2+1$$
$$= \sum_{i=0}^{\log n} 2^i = \frac{x^{(n+1)}-1}{x-1} = \frac{2^{(\log n+1)}-1}{2-1}$$
$$= \frac{2^{(\log n)}+2-1}{2-1} = \frac{n^{(\log 2)}+2-1}{2-1} = \frac{n+2-1}{2-1} = \Theta(n)$$

Example-5: Solving Recurrence

$f(n) = f(n/2) + n$

$= f(n/4) + n/2 + n$

$= f(n/8) + n/4 + n/2 + n$

$= f(n/16) + n/8 + n/4 + n/2 + n$

$\ldots$

$= f(n/n) + 2 + 4 + \ldots + n/8 + n/4 + n/2 + n$

$= 1 + 2 + 4 + \ldots + n/8 + n/4 + n/2 + n$

$$= \sum_{i=0}^{\log n} \frac{n}{2^i} = n \sum_{i=0}^{\log n} \frac{1}{2^i}$$

this is a decreasing series, so

$$\sum_{i=0}^{\infty} c^i = \frac{1}{1-c} \quad \text{for } |c| < 1$$

using this we have $c = 1/2$

$$n \sum_{i=0}^{\infty} \frac{1}{2^i} = n \frac{1}{1-c} = n \frac{1}{1-(1/2)} = n \frac{1}{1/2} = 2n$$

$T(n) \le 2n \in \Theta(n)$

Selection Problem

- Suppose you have a series of numbers
  - 5, 7, 2, 10, 8, 15, 21, 37, 41
- How many numbers are smaller than 10
- Rank of a number is the position of number in sorted sequence
- To find the rank of all number we can write $O(n^2)$ algorithm.
- We can sort the numbers

Pos	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Num	2	5	7	8	10	15	21	37	41

- Sorting takes $O(n \log n)$.
- Left to right scanning of array is $O(n)$.

Median

- Median is the middle rank number in a list of number.
- Median is of particular interest in statistics.
- Medians are useful as a measure of central tendency of a set especially when the distribution of values is highly skewed.
- For example, the median income in a community is more meaningful measure than average.

Median (cont'd)

- Suppose 7 households have monthly incomes 5000, 7000, 2000, 10000, 8000, 15000 and 16000
- In sorted order incomes are 2000, 5000, 7000, 8000, 10000, 15000, 16000
- The Median income is 8000; median is element with rank 4: $(7+1)/2=4$
- The average income is 9000

Median (cont'd)

- Suppose the income 16000 goes up to 450,000
- The median is still 8000, but the average goes up to 71000
- Clearly the average is not good representative of the majority of income levels.

Medians & Selection -Sieve Technique

- We will use a variation of divide and conquer strategy called sieve technique.
- In divide & conquer, we break the problem into a small number of smaller subproblems which we solve recursively.
- In Selection Problem we are looking for an item.
- We will divide the problem into subproblems.
- However we will discard those smaller subproblems for which we determine that they do not contain the desired answer (the variation).

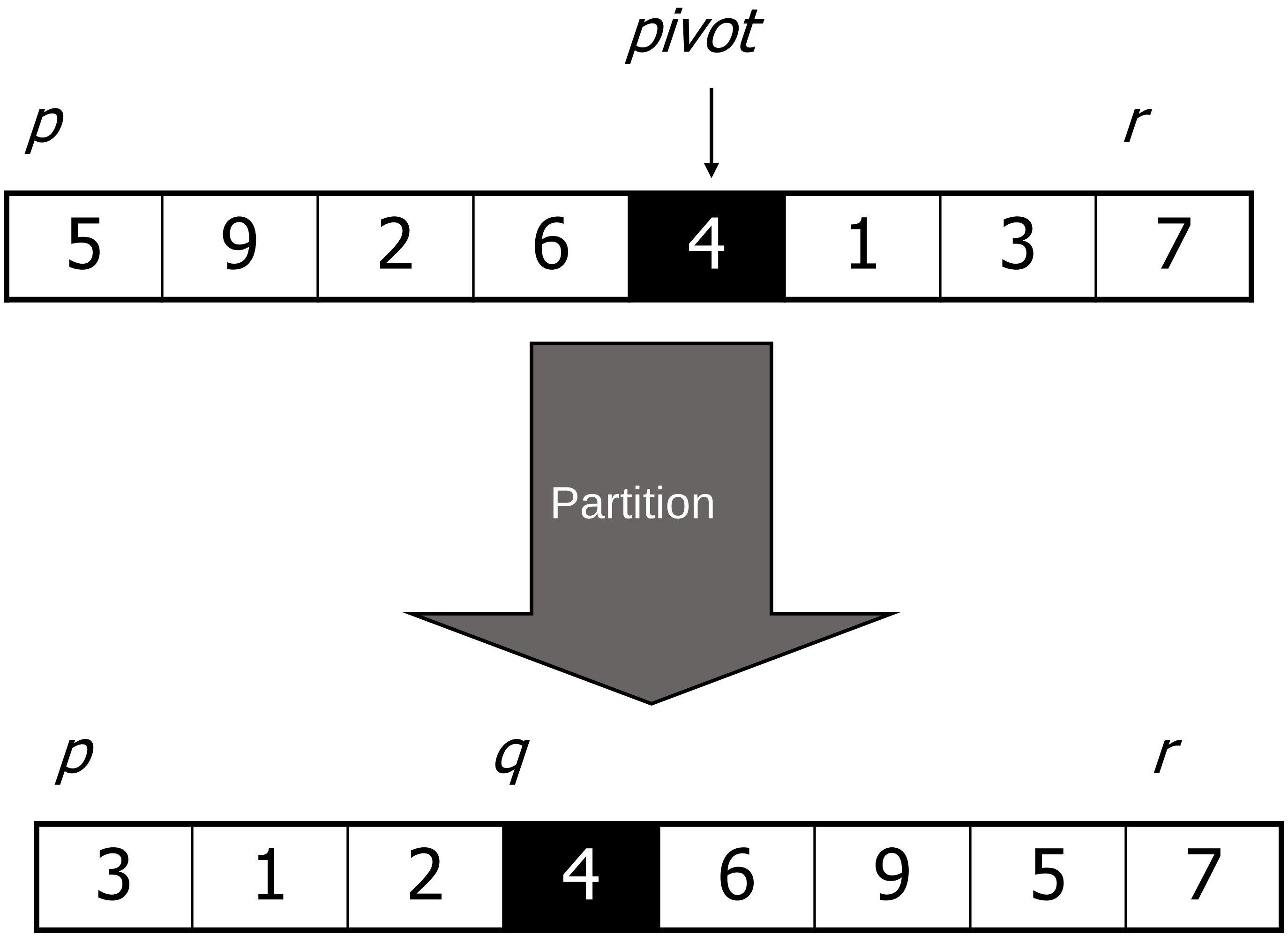
Medians & Selection -Sieve Technique (cont'd)

- Here is how the sieve technique will be applied to the selection problem
  - We will begin with the given array A[1..n]
  - We will pick an item from the array, called the pivot element which we will denote by x.
  - For now just think of pivot element as a random element of A.

Medians & Selection -Sieve Technique (cont'd)

- We partition A into three parts
  1. A[q] contains the pivot element x.
  2. A[1..q-1] will contain all the elements that are less than x
  3. A[q+1..n] will contain all the elements that are greater than x
- Within each sub-array the items may appear in any order.

Medians & Selection -Sieve Technique (cont'd)



Medians & Selection -Sieve Technique (cont'd)

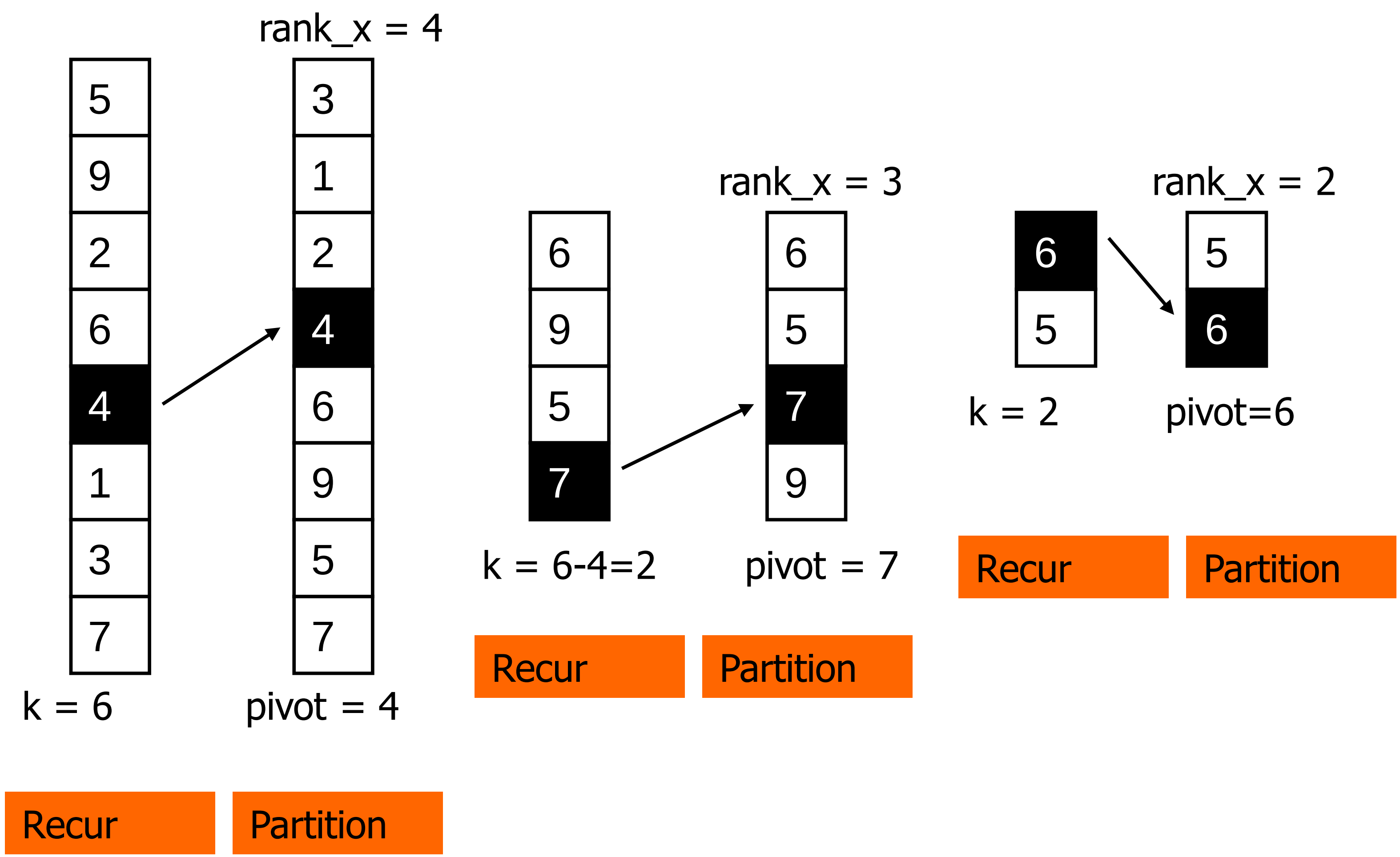
- The rank of the pivot x is
  - $q - p + 1$ in A[p..r]
- Let $rank\_x = q - p + 1$
- If $k = rank\_x$ then the pivot is k^{th} smallest.
- If $k < rank\_x$ then search A[p..q-1] recursively
- If $k > rank\_x$ then search A[q+1..r] recursively. Find element of rank (k-q) because we eliminated q smaller elements in A.

SELECT Algorithm

```
SELECT(array A, int p, int r, int k)
   if (p = r) then return A[p]
   else x ← CHOOSE_PIVOT(A, p, r)
   q ← PARTITION(A, p, r, x)
   rank_x ← q - p + 1
   if k = rank_x then return x
   if k < rank_x then return SELECT(A, p, q−1, k)
   else return SELECT(A, q+1, r, k−q)
```

Example: SELECT Algorithm

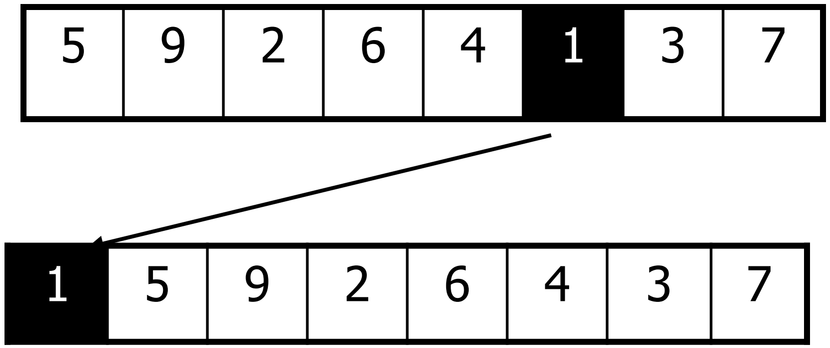
- Select the 6th smallest element of the set $\{5, 9, 2, 6, 4, 1, 3, 7\}$



Analysis of SELECT Algorithm

- For the moment, we will assume that partitioning and pivot both take $\Theta(n)$ time
- How many elements do we eliminate each time?
- If x is the smallest or the largest then we may only succeed in eliminating one element

Analysis of SELECT Algorithm (cont'd)



Pivot is 1

After partitioning

- Ideally, x should have a rank that is neither too large or too small

Analysis of SELECT Algorithm (cont'd)

- Suppose we are able too choose a pivot that causes exactly half of the array to be eliminated in each phase
- This means that we recurse on the remaining n/2 elements
- This leads to the following recurrence

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + n & \text{otherwise} \end{cases}$$

Analysis of SELECT Algorithm (cont'd)

- If we expand this recurrence we get

$$T(n) = n + \frac{n}{2} + \frac{n}{4} + \dots$$

$$\leq \sum_{i=0}^{\infty} \frac{n}{2^i}$$

$$= n \sum_{i=0}^{\infty} \frac{1}{2^i}$$

Analysis of SELECT Algorithm (cont'd)

- Recall the formula for infinite geometric series; for any $|c| < 1$,

$$\sum_{i=0}^{\infty} c^i = \frac{1}{1-c}$$

using this we have

$$n\sum_{i=0}^{\infty} \frac{1}{2^i} = n\frac{1}{1-c} = n\frac{1}{1-(1/2)} = n\frac{1}{1/2} = 2n$$

$$T(n) \le 2n \in \Theta(n)$$

Analysis of SELECT Algorithm (cont'd)

- Lets think about how we ended up with a $\Theta(n)$ algorithm for selection
- Normally a $\Theta(n)$ algorithm would make a single or perhaps a constant number of passes of the data set
- In this algorithm we make a number of passes. In fact it could be as many as log n
- However, because we eliminate a constant fraction of the array with each phase, we get the convergent geometric series in the analysis.