Maximum Stock Profit

DAC Template

Given a problem of size *n:*

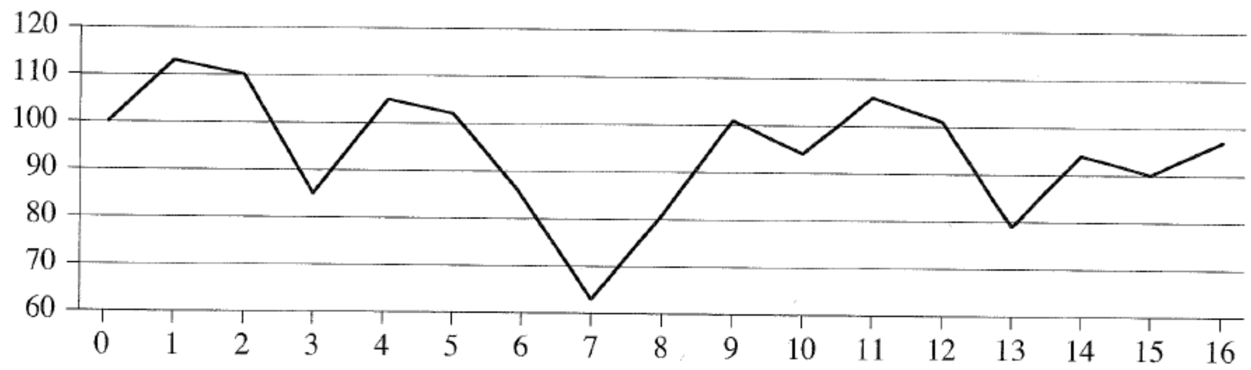
1. **Divide** the problem into *k* subproblems of size *n/k* each
2. **Conquer** by solving each subproblem independently
3. **Combine** the *k* solutions to subproblems into a solution to the original problem

Time? $T(n) = k \cdot T(n/k) + d(n) + c(n)$

Example: For MergeSort, $k = 2, d(n) \in O(1)$, $c(n) \in \Theta(n)$.

Maximum Stock Profit Revisited

- Given a historical list of daily stock prices, when was the best time to buy and sell in order to maximize profit?

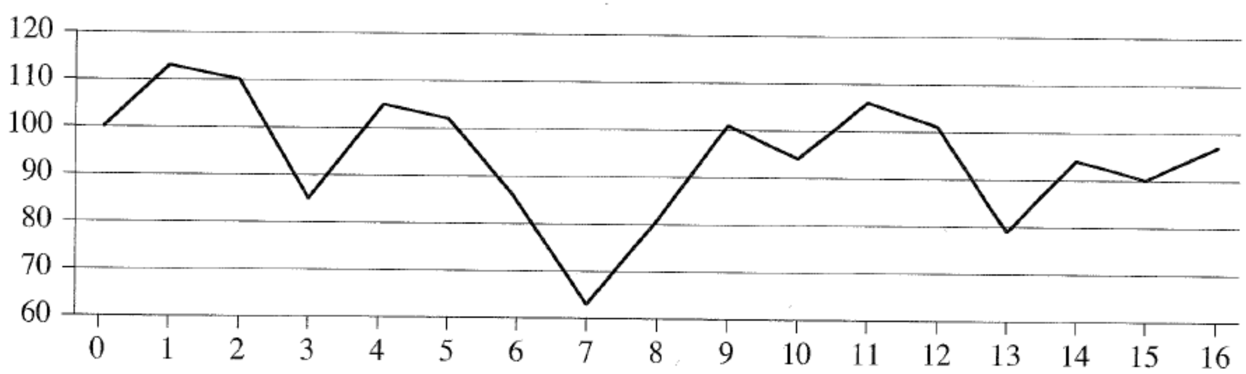


- Recall that we had a $\Theta(n^2)$ brute force algorithm that considered all feasible combinations of buying and selling times.
- Can we do better?

A Transformation

- What if, instead, you work with the sequence of daily price changes?

Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7



- *New goal:* Find a *contiguous* subarray whose sum is largest.

MaxSum Subarray

Algorithm 1: For each subarray $A[i..j]$, find the net change and keep the maximum.

```
def MSS1(A):
    best = 0
    n = len(A)
    for i in range(n):
        for j in range(i,n):
            sum = 0
            for k in range(i,j+1):
                sum = sum + A[k]
            if sum > best:
                best = sum
    return best
```

n n^2 n $*$

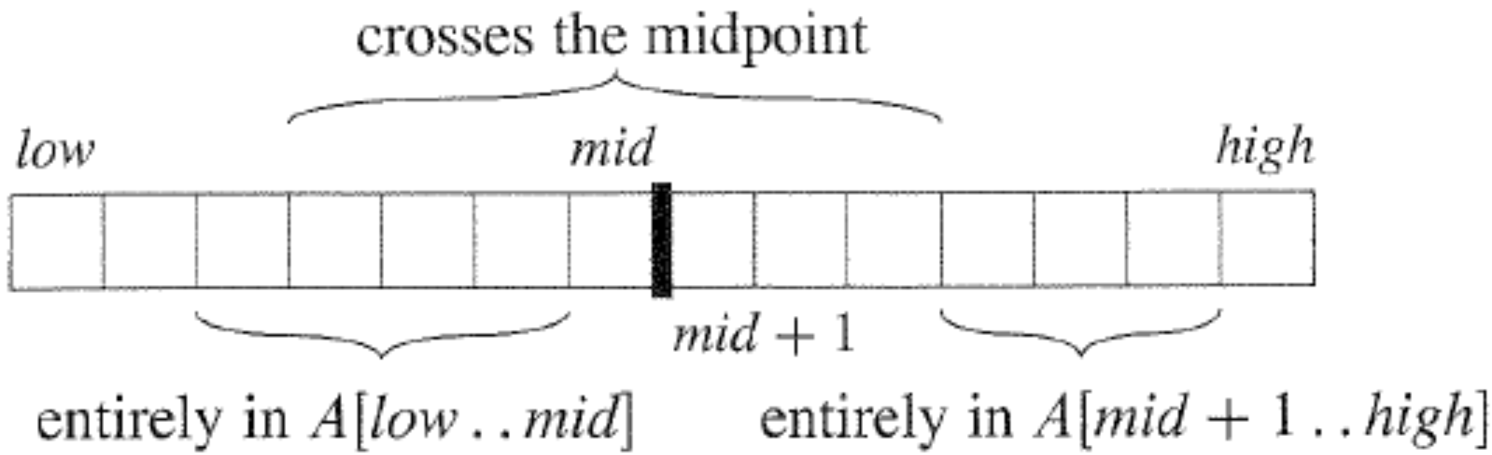
$$T(n) = \Theta(n^3)$$

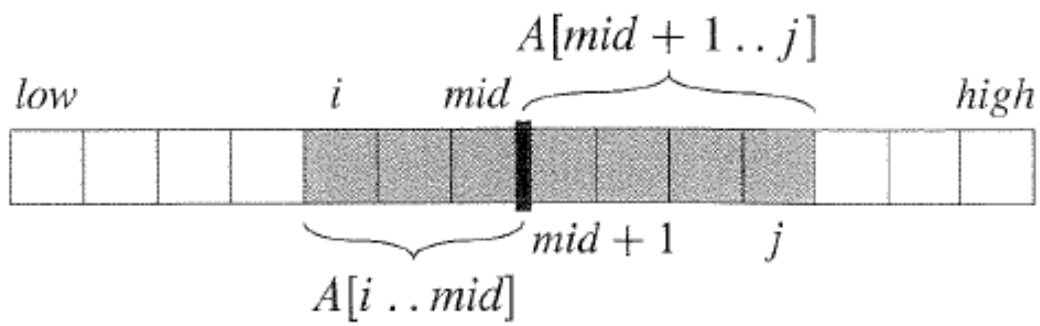
Algorithm 2

```
def MSS2(A):
    best,ffrom,to = 0,0,-1
    n = len(A)
    for i in range(n):
        sum = 0
        for j in range(i,n):
            sum = sum + A[j]
            if sum > best:
                best,ffrom,to = sum,i,j
    return best
```

$$T(n) = \Theta(n^2)$$

Algorithm 3: DAC





```python
def MSSDAC(A,low=0,high=None):
    if high==None: high = len(A)-1
    # Base case
    if low == high:
        if A[low]>0: return A[low]
        else: return 0
    # Divide
    mid = (low+high)//2
    # Conquer
    maxLeft = MSSDAC(A, low, mid)
    maxRight = MSSDAC(A, mid+1, high)
    # Combine
    maxLeft2Center = left2Center = 0
    for i in range(mid,low-1,-1):
        left2Center += A[i]
        maxLeft2Center = max(left2Center,maxLeft2Center)
    maxRight2Center = right2Center = 0
    for i in range(mid+1,high+1):
        right2Center += A[i]
        maxRight2Center = max(right2Center,maxRight2Center)
    return max(maxLeft,maxRight,maxLeft2Center+maxRight2Center)
```

$$T(n) = 2T(n/2) + n \in \Theta(n \log n)$$

END

Matrix Multiplication

Matrix Multiplication Revisited

- Given two $n \times n$ matrices A and B, find $C = A \times B$
- Recall: C is also $n \times n$ and

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Example: $n = 2$

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

- We already have a $\Theta(n^3)$ "brute force" algorithm (in terms of the input size $N = 2n^2$, $T(N) \in \Theta(N\sqrt{N})$)

A DAC Solution

- Partition each $n \times n$ matrix into four $n/2 \times n/2$ submatrices.

$$A = \left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right), \quad B = \left(\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right), \quad C = \left(\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right)$$

$$\begin{aligned} C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21} , \\ C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} , \\ C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} , \\ C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22} . \end{aligned}$$

Recurrence? $T(n) = 8T(n/2) + n^2$

A DAC Solution

```
SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)
n = A.rows
let C be a new n × n matrix
if n == 1
    c11 = a11 · b11
else partition A, B, and C as in equations (4.9)
    C11 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B11)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B21)
    C12 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B12)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B22)
    C21 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B11)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B21)
    C22 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B12)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B22)
return C
```

Time? $T(n) = 8T(n/2) + n^2 \in \Theta(n^3)$

$$T(n) = n^2 + 8 \cdot T(n/2)$$

$$= n^2 + 8[(n/2)^2 + 8 \cdot T(n/4)] = n^2 + 2n^2 + 8^2 \cdot T(n/4)$$

$$= n^2 + 2n^2 + 8^2[(n/4)^2 + 8 \cdot T(n/8)]$$

$$= n^2 + 2n^2 + 2^2n^2 + 8^3 \cdot T(n/2^3)$$

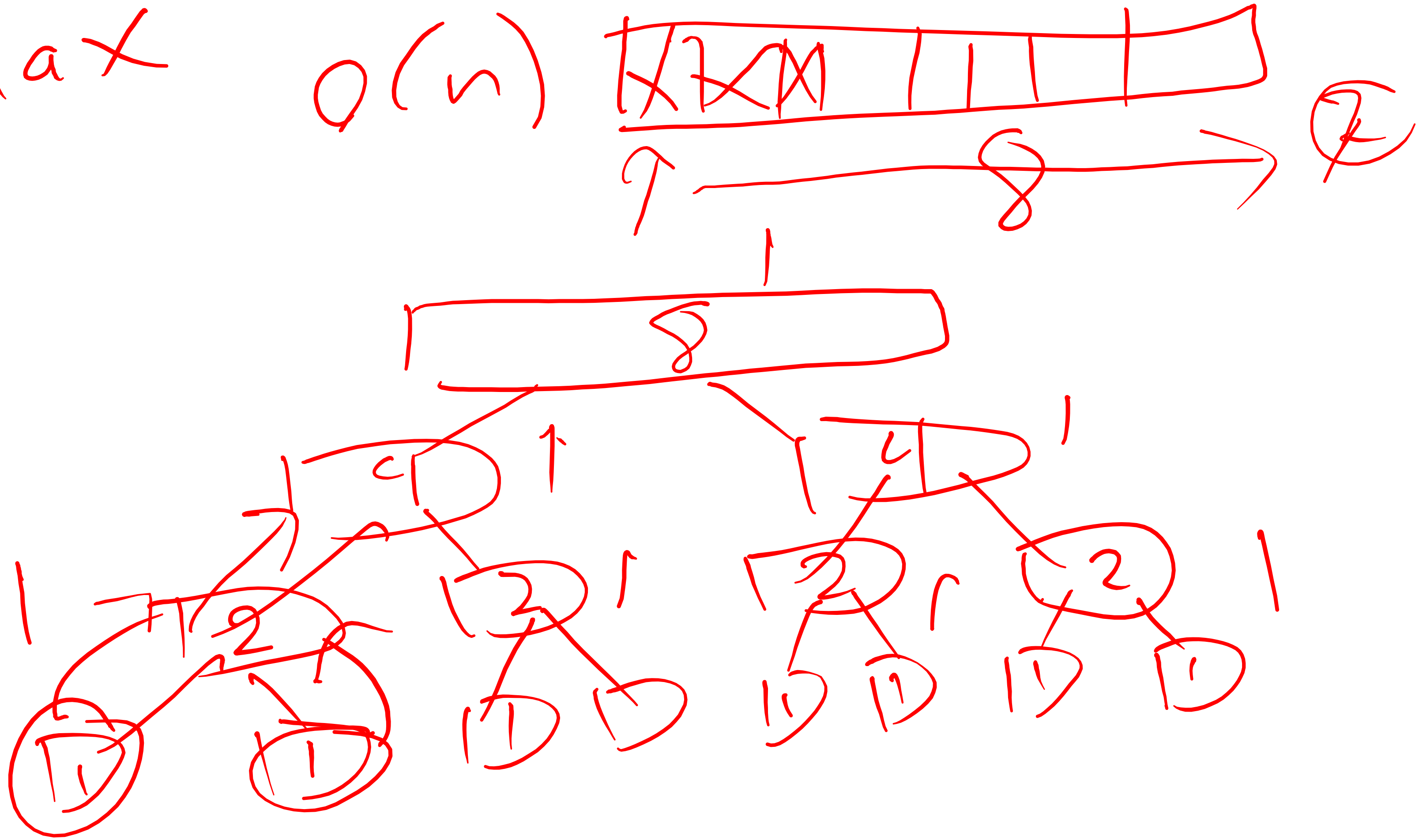
$$= n^2 + 2n^2 + 2^2n^2 + \cdots + 2^{k-1}n^2 + 8^k \cdot T(n/2^k)$$

$$= n^2 \left(1 + 2 + \cdots + 2^{k-1}\right) + 2^{3k} \cdot T(n/2^k)$$

$$= n^2 \left(2^k - 1\right) + 2^{3k} \cdot T(n/2^k)$$ **Stop when $\boldsymbol{n = 2^k}$**

$$= n^2 \left(n - 1\right) + n^3 \cdot T(1) \qquad \in \Theta(n^3)$$

max $O(n)$



A Different DAC Solution (Strassen, 1968)

$$
\begin{aligned}
S_1 &= B_{12} - B_{22}\,, & S_6 &= B_{11} + B_{22}\,,\\
S_2 &= A_{11} + A_{12}\,, & S_7 &= A_{12} - A_{22}\,,\\
S_3 &= A_{21} + A_{22}\,, & S_8 &= B_{21} + B_{22}\,,\\
S_4 &= B_{21} - B_{11}\,, & S_9 &= A_{11} - A_{21}\,,\\
S_5 &= A_{11} + A_{22}\,, & S_{10} &= B_{11} + B_{12}\,.
\end{aligned}
$$

$$
\begin{aligned}
P_1 &= A_{11} \cdot S_1 &&= A_{11} \cdot B_{12} - A_{11} \cdot B_{22}\,,\\
P_2 &= S_2 \cdot B_{22} &&= A_{11} \cdot B_{22} + A_{12} \cdot B_{22}\,,\\
P_3 &= S_3 \cdot B_{11} &&= A_{21} \cdot B_{11} + A_{22} \cdot B_{11}\,,\\
P_4 &= A_{22} \cdot S_4 &&= A_{22} \cdot B_{21} - A_{22} \cdot B_{11}\,,\\
P_5 &= S_5 \cdot S_6 &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22}\,,\\
P_6 &= S_7 \cdot S_8 &&= A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22}\,,\\
P_7 &= S_9 \cdot S_{10} &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}\,.
\end{aligned}
$$

$$
\begin{aligned}
C_{11} &= P_5 + P_4 - P_2 + P_6\\
C_{12} &= P_1 + P_2\\
C_{21} &= P_3 + P_4\\
C_{22} &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$

Strassen

$$C_{11} = P_5 + P_4 - P_2 + P_6$$

$$C_{12} = P_1 + P_2$$

$$C_{21} = P_3 + P_4$$

$$C_{22} = P_5 + P_1 - P_3 - P_7$$

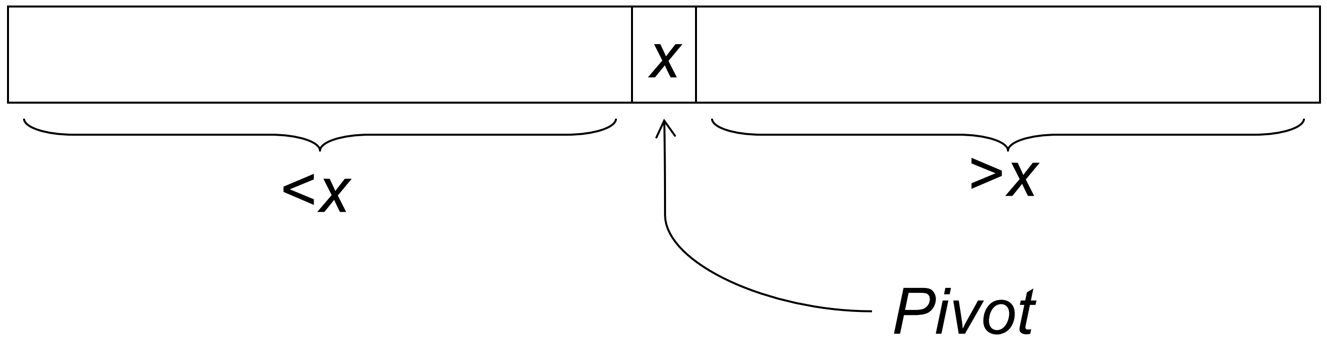
Recurrence? $T(n) = 7T(n/2) + n^2 = n^{\log_2 7}$

END

Quicksort

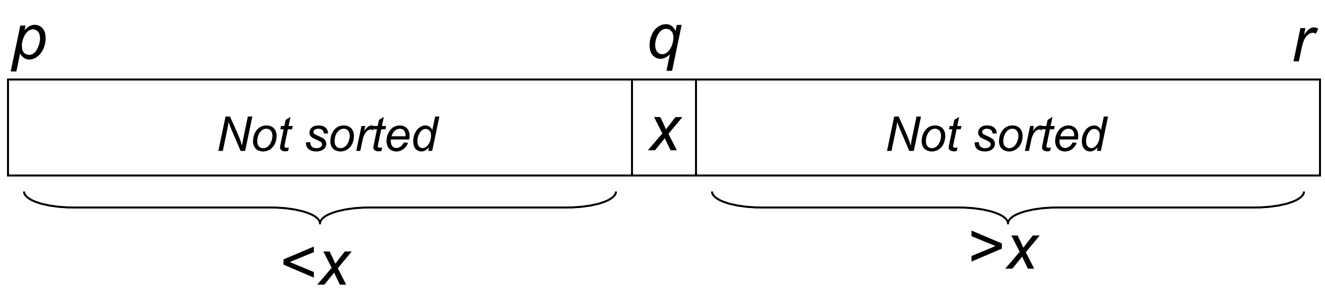
Quicksort

- Worst case is $\Theta(n^2)$
- Best case is $\Theta(n \log n)$
- Average case is $\Theta(n \log n)$
- Constant hidden in Θ-notation is small
- Sorts in place
- Divide and conquer
  - Based on linear time partition algorithm (a clever divide)



Quicksort

1. *Divide:* Rearrange $A[p..r]$ into three parts—$A[p..q-1]$, $A[q]$, and $A[q+1..r]$—such that each element of the first part is $< A[q]$ and each element of the third part is $> A[q]$.
2. *Conquer:* Recursively sort the two unsorted parts.
3. *Combine:* Not needed!



Example

								x	
14	27	3	75	41	4	13	65	23	54

Partition

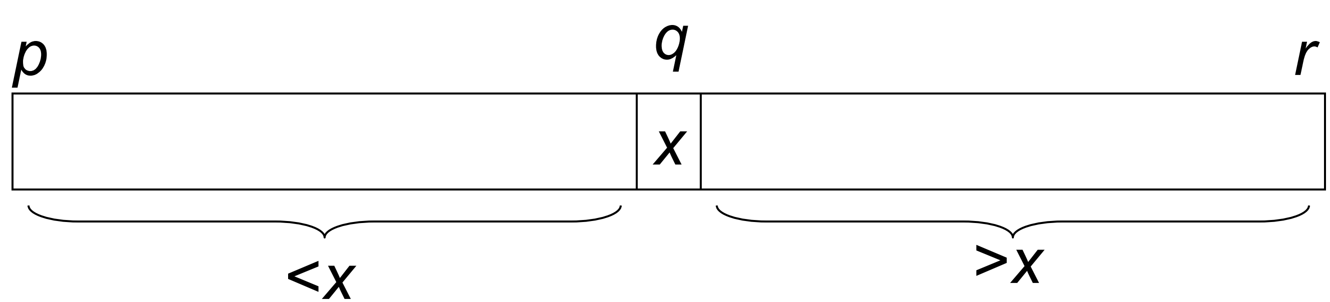
14	3	4	13	23	27	75	41	65	54

Sort recursively

3	4	13	14	23	27	41	54	65	75

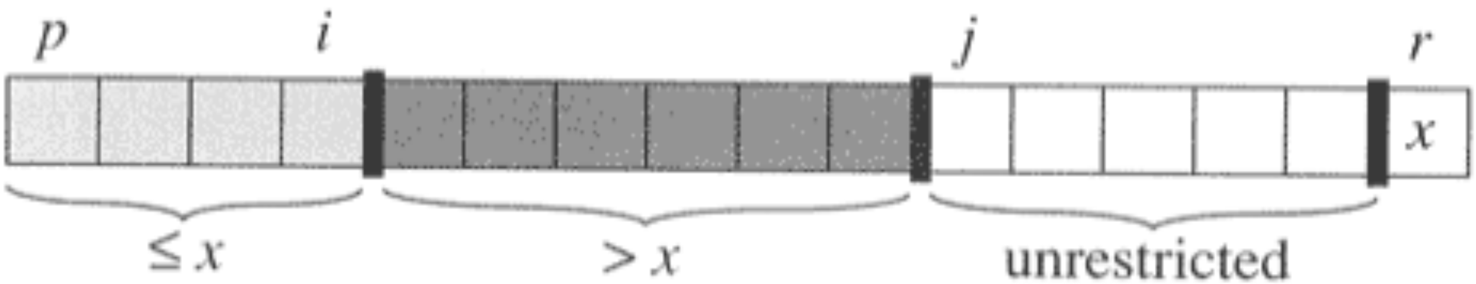
Quicksort

```
QUICKSORT(A, p, r)
if p < r
    then q ← PARTITION(A, p, r)
         QUICKSORT(A, p, q − 1)
         QUICKSORT(A, q + 1, r)
```



Partition

- Choose pivot $x = A[r]$
- Scan array once from left to right
- During partition, array consists of four parts
  - Portion already scanned $A[p..j-1]$ is partitioned into two parts—$A[p..i]$ and $A[i+1..j-1]$—of elements smaller and bigger than x, respectively



Partition

```
PARTITION(A, p, r)
x ← A[r]
i ← p − 1
for j ← p to r − 1
     do if A[j] ≤ x
           then i ← i + 1
                exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return i + 1
```

Runs in $\Theta(n)$ time (where $n = r - p + 1$)

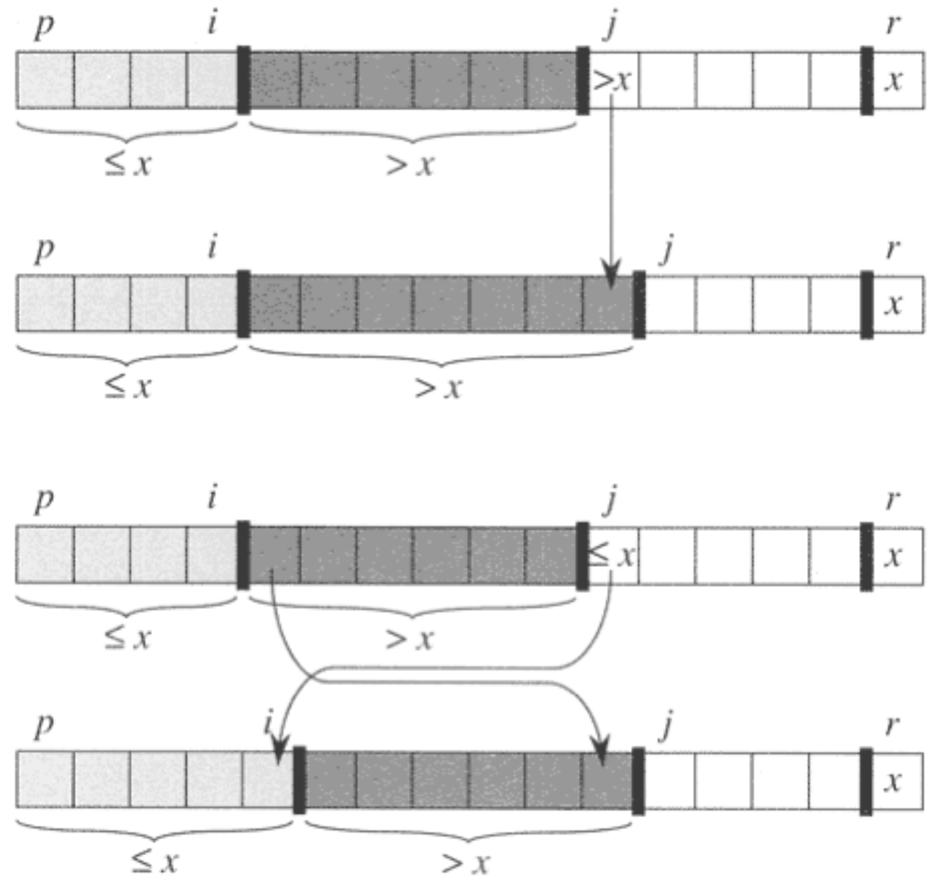
Processing the Next Element

Two cases:

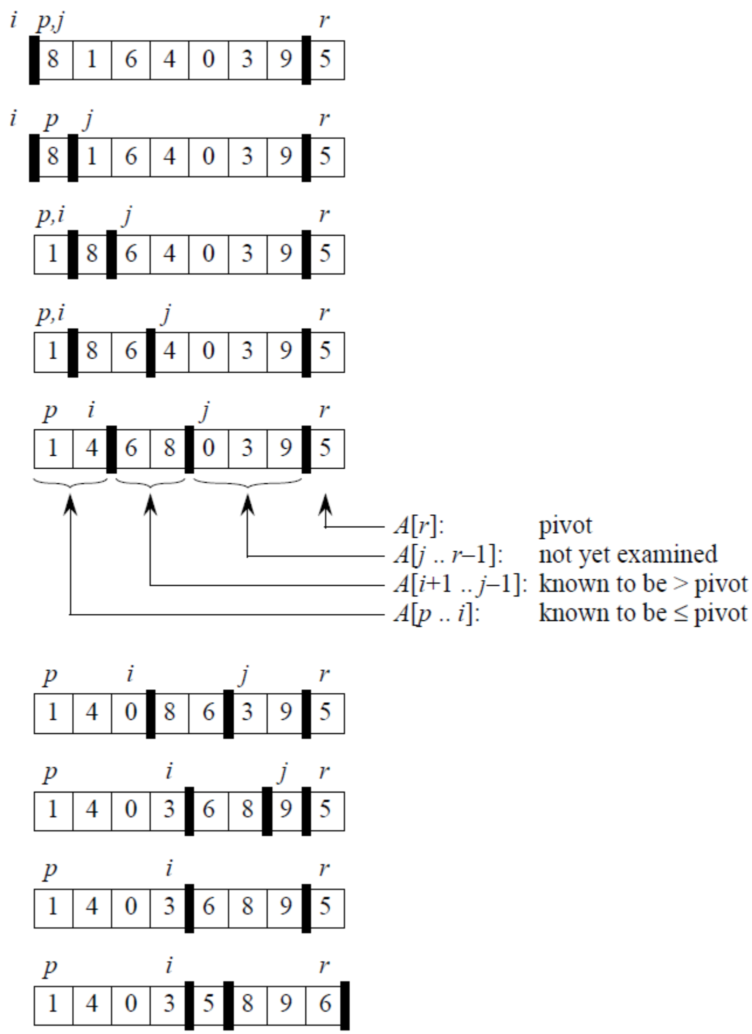
a. $A[j] > x$
b. $A[j] \leq x$

(a)

(b)



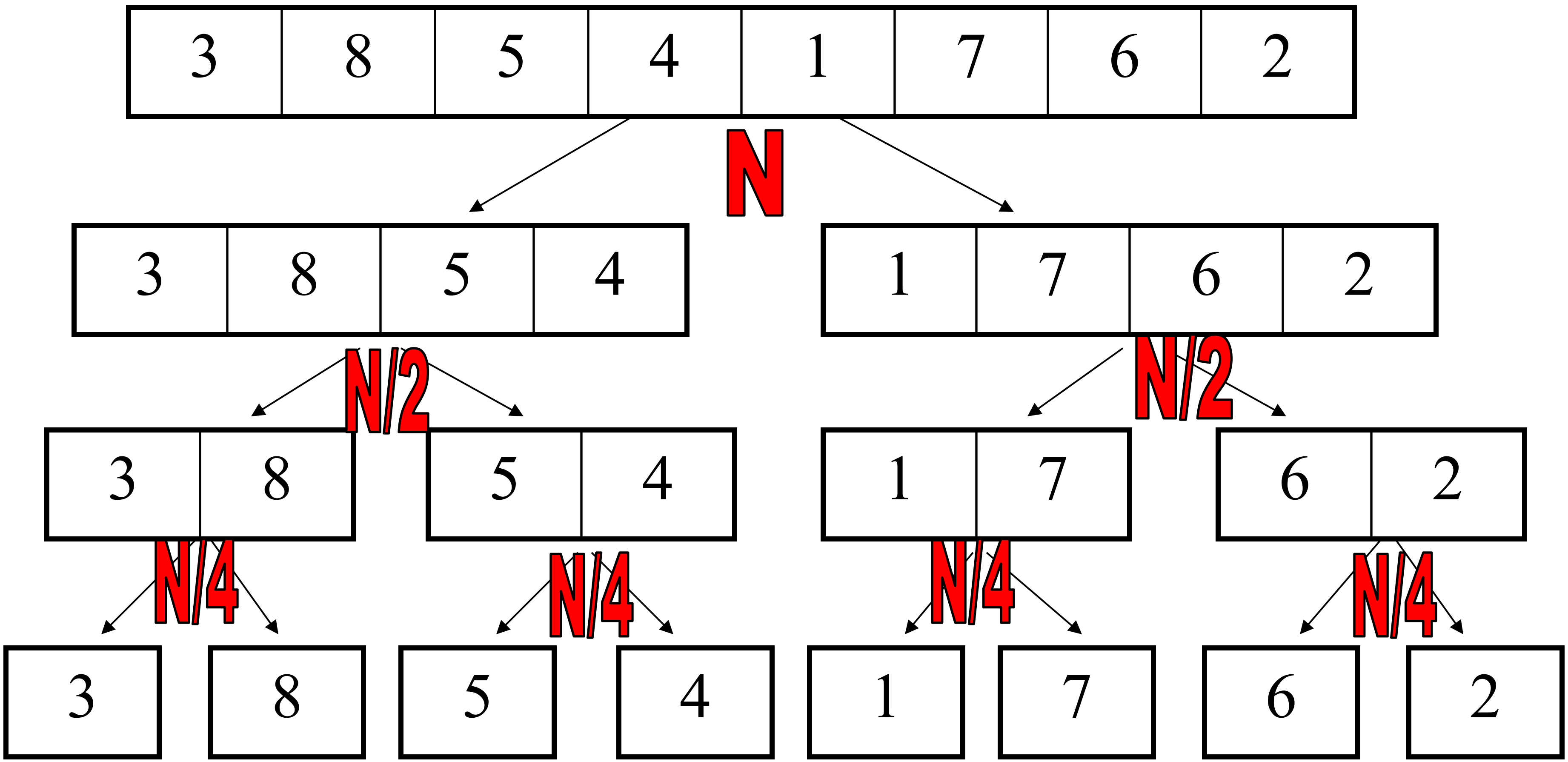
Example




```
PARTITION(A, p, r)
x ← A[r]
i ← p − 1
for j ← p to r − 1
    do if A[j] ≤ x
        then i ← i + 1
            exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return i + 1
```

Running Time

- Recall for merge sort: Divide: O(1), Merge: O(n)

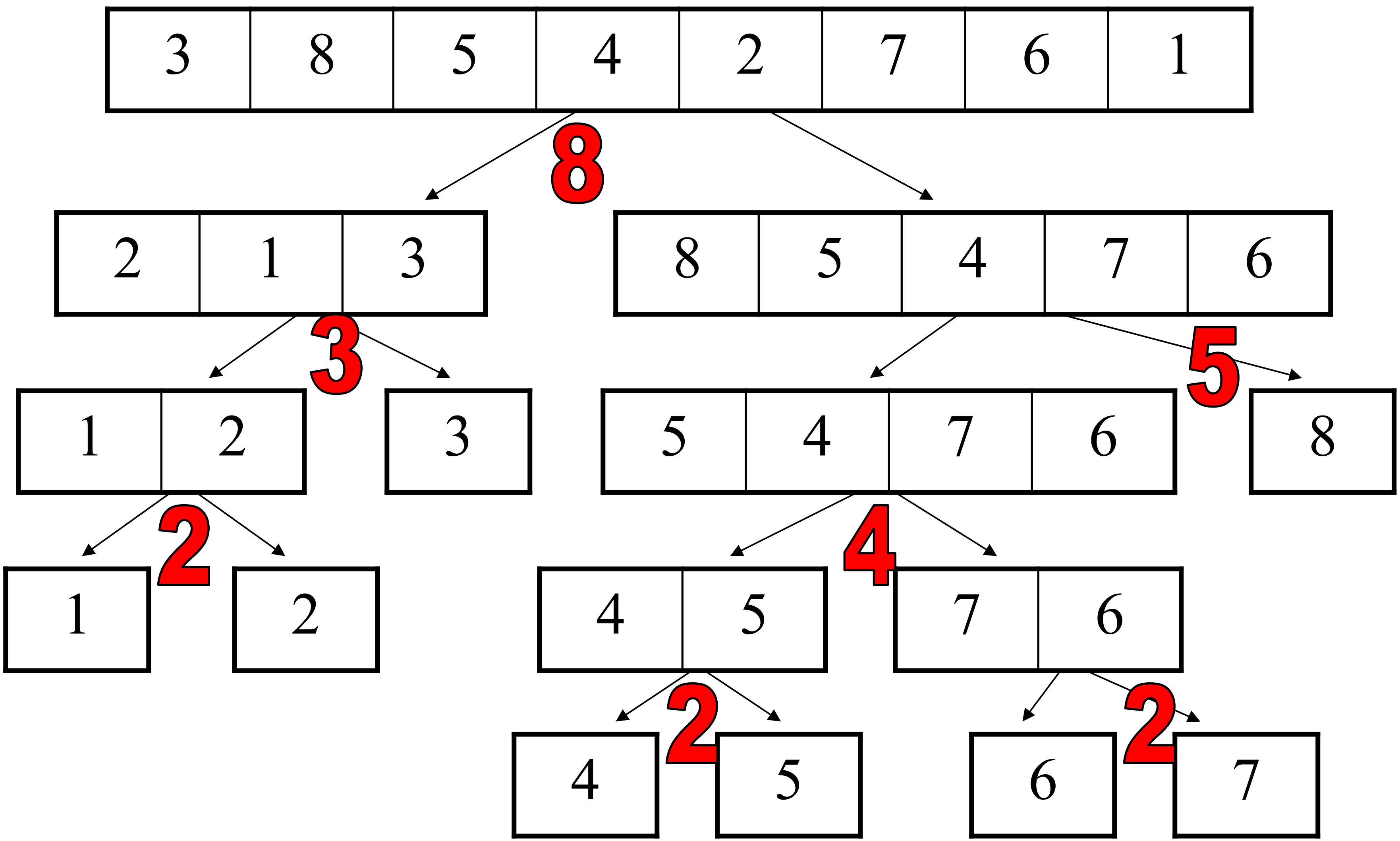


Running Time

- Merge had n amount of work at each level
- Merge had log(n) levels
- Multiply the two since it has the same amount at each level giving O(n log(n))

- Try this same approach for analysis on Quick
    - Partition: O(n)
    - Combine: O(1)

Running Time



Running Time

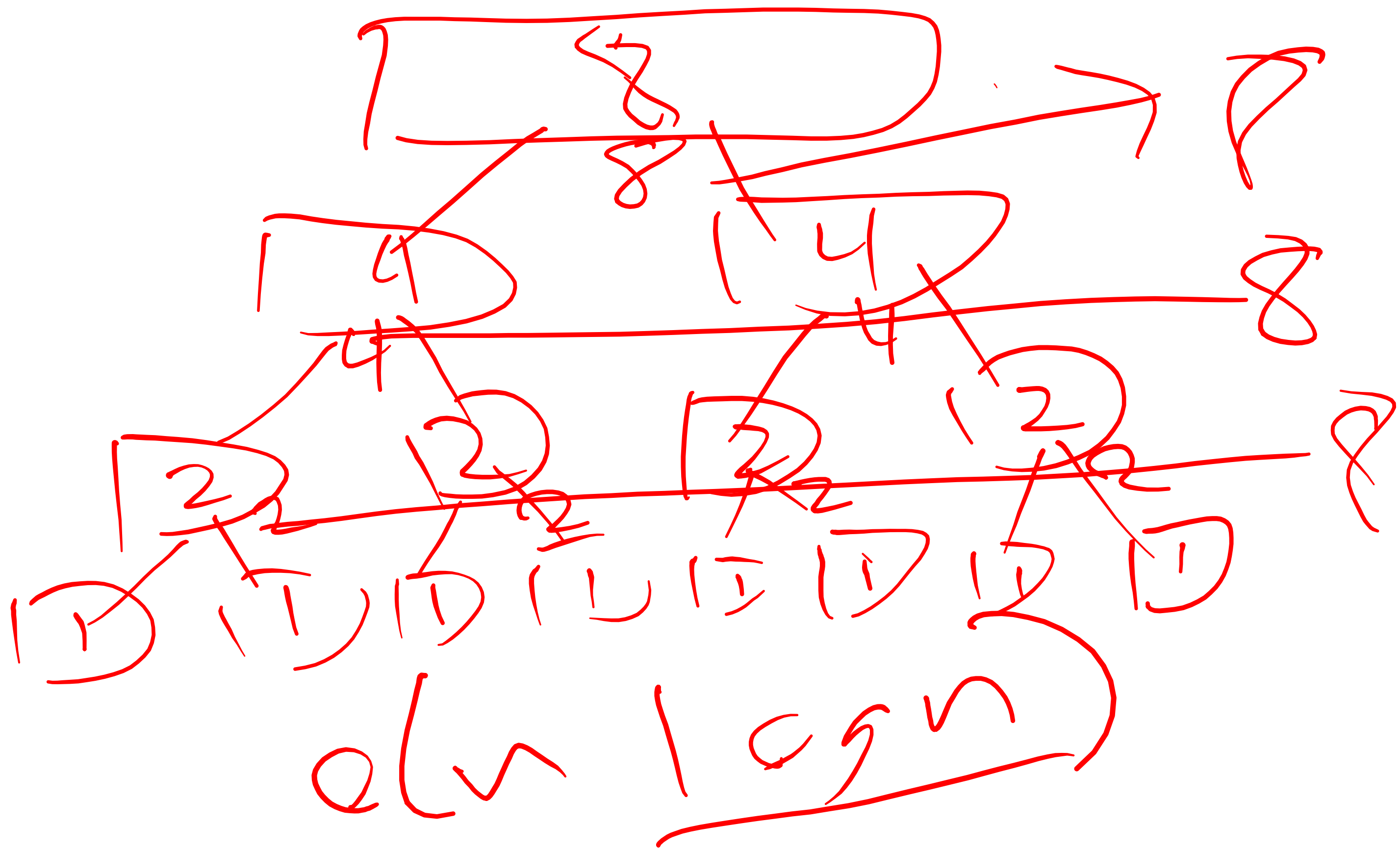
- There is not the same amount of work at each level
- Cannot simply multiply number of levels by work

- The choice of partition is an important factor in how much work is necessary

- What are the best and worst cases?
  - What are their associated runtimes?

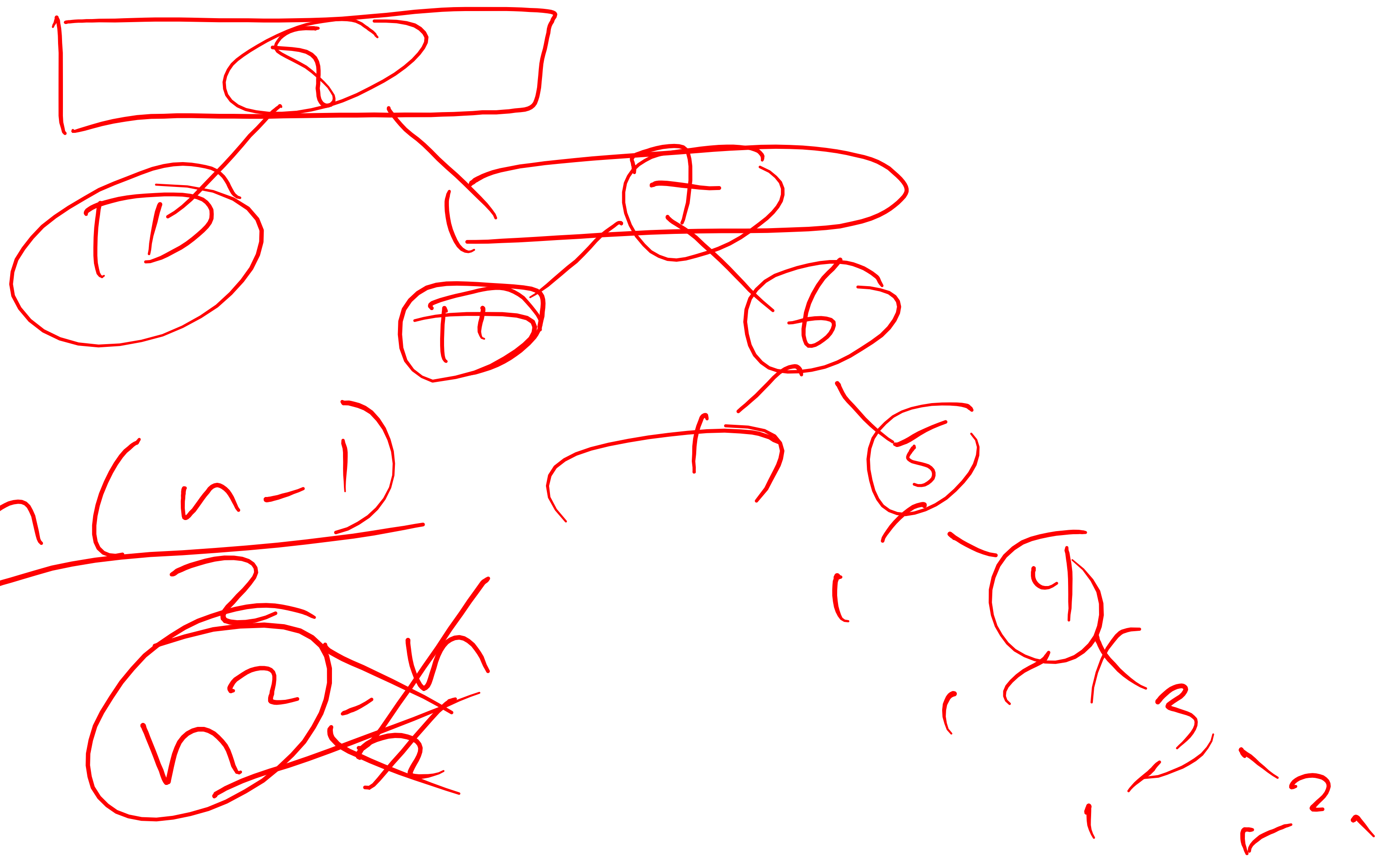
Running Time: Best Case

- Best case is when we divide evenly like merge



Running Time: Worst Case

- Worst case is when we partition very unevenly

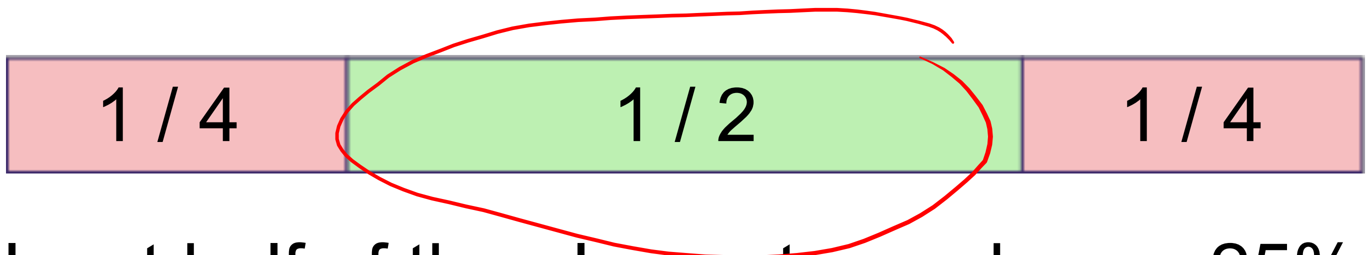


$$\frac{n(n-1)}{2}$$

$$n^2 \cancel{-\frac{n}{2}}$$

Average Informal Analysis

- Always choose the pivot at random.
- Consider the (hypothetically) sorted array.



- At least half of the elements produce a 25%:75% split or better.
- $\Rightarrow$ roughly every other iteration, the largest partition contains at most 3/4 of the data.
- Randomized quicksort runs in $\Theta(n \log n)$ with very high probability.

How Do You Make All Pivots Equally Likely?

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

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

END

Order Statistics

Order Statistics

- Given a list A of n elements, a ***percentile*** is a value below which a given percentage of observations in A falls.

  Example: the 30^{th} percentile is the value below which 30% of the observations may be found

- The j^{th} order statistic of a list of n elements is the j^{th} smallest element of the list (i.e., element of rank j).
  - The p^{th} percentile corresponds to the $pn/100$ order statistics

  Example: $S = \{8, 40, 30, 17, 31, 4, 50, 42, 41, 27, 19\}$

  $i = 3 \Rightarrow$ third smallest element is 17

  $i = 1 \Rightarrow$ smallest element (minimum) is 4

  $i = n \Rightarrow$ largest element (maximum) is 50

  $i = \lfloor (n + 1)/2 \rfloor$?

First-Order Statistic

```
MINIMUM(A)
1  min ← A[1]
2  for i ← 2 to length[A]
3        do if min > A[i]
4              then min ← A[i]
5  return min
```

$\boxed{n}$

n^2

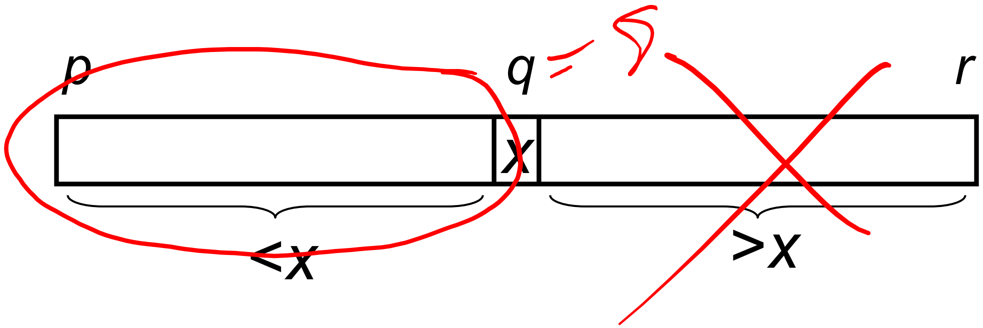
$n \log n$

- Cost = number of comparisons (line 3)
- Can we do better?

*i*th-Order Statistic

i = 5

- Can *partition* help?



- What is the relation between *i*, *p*, and *q*?
- Simple recursive algorithm

$$T(n) = T(k) + \Theta(n), \text{ for some } k < n$$

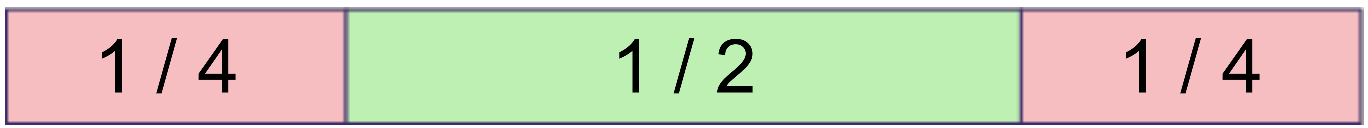
A Randomized Algorithm

```
RANDOMIZED-SELECT(A, p, r, i)
if p = r
    then return A[p]
q ← RANDOMIZED-PARTITION(A, p, r)
k ← q − p + 1
if i = k          ▷ the pivot value is the answer
    then return A[q]
elseif i < k
    then return RANDOMIZED-SELECT(A, p, q − 1, i)
else return RANDOMIZED-SELECT(A, q + 1, r, i − k)
```

- $T(n)$ = time for randomized select with n elements
  - $T(n)$ is a random variable
  - $E[T(n)]$?
  - Worst case?

Informal Analysis

- Choose the pivot at random.
- Consider the sorted array.



- At least half of the elements produce a 25%:75% split or better.
- In roughly every other iteration, the array reduces by at least 25%.

$$n \rightarrow \frac{3n}{4} \rightarrow \frac{9n}{16} \rightarrow \cdots \rightarrow \left(\frac{3}{4}\right)^k n$$

Formal Analysis

- $T(n)$ is our random variable whose expected value we wish to find.
- Element z_k of rank k, chosen with probability $1/n$
- Since $E(T(n))$ is monotonically increasing:

$$E[T(n)] \leq \frac{1}{n} \sum_{k=1}^{n} E[T(\max(k-1, n-k)) + \Theta(n)]$$

$$\max(k-1, n-k) = \begin{cases} k-1 & \text{if } k > \lceil n/2 \rceil \\ n-k & \text{if } k \leq \lceil n/2 \rceil \end{cases}$$

$$E[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[T(k)] + \Theta(n)$$

Claim: E[T(n)] ≤ cn

- Base case?
  - Let *k* satisfy $T(n) \le cn$, if $n \le d$ (will find *d* later)
- Inductive step

$$E[T(n)] \le \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[T(k)] + an$$

$$\le \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + an$$

$$= \frac{2c}{n} \left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) + an$$

$$
\begin{aligned}
E[T(n)] &\le \frac{2c}{n}\left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2\rfloor - 1} k\right) + an \\
&= \frac{2c}{n}\left(\frac{n(n-1)}{2} - \frac{\lfloor n/2\rfloor(\lfloor n/2\rfloor - 1)}{2}\right) + an \\
&\le \frac{2c}{n}\left(\frac{n(n-1)}{2} - \frac{(n/2-1)(n/2-2)}{2}\right) + an \\
&= \frac{c}{n}\left(\frac{3n^2}{4} + \frac{n}{2} - 2\right) + an \\
&= c\left(\frac{3n}{4} + \frac{1}{2} - \frac{2}{n}\right) + an \\
&\le \frac{3cn}{4} + \frac{c}{2} + an \\
&= cn - \left(\frac{cn}{4} - \frac{c}{2} - an\right) \le cn \text{ if } c > 4a, n \ge \frac{2c}{c-4a}
\end{aligned}
$$

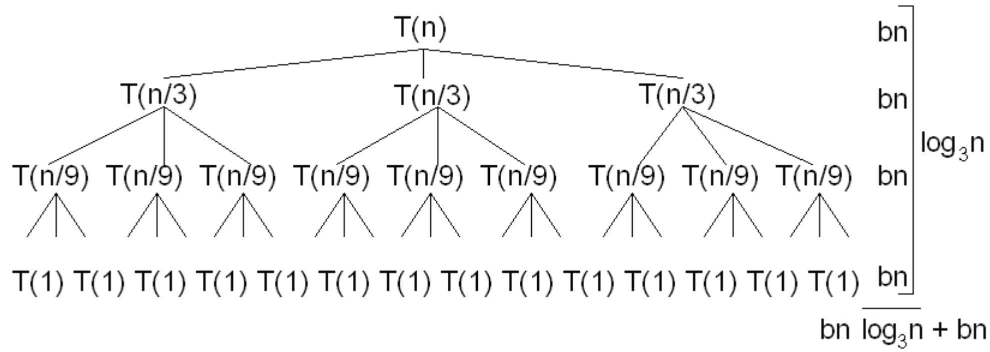
END

Master Method

Solving Recurrences

- Iteration/recursion trees

$$
\begin{aligned}
T(n) &= n + 3T(n/3) \\
&= n + 3(n/3 + 3T(n/9)) \\
&= 2n + 9T(n/9) \\
&= 3n + 27T(n/27) \\
&\vdots \\
&= kn + 3^k T(n/3^k)
\end{aligned}
$$



- Substitution (induction)
- Master theorem

Substitution

- Prove $T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n)$
  - Will show $T(n) \le bn \log n$, $n \ge 2$ (upper bound)
  - Can assume
    - $T(n) \le 2T(n/2) + cn$
    - $T(k) \le bk \log k$, for $2 \le k < n$
- **Proof**

$$\begin{aligned}
T(n) &\le 2T(n/2) + cn \\
&\le 2\left(b\frac{n}{2}\log\frac{n}{2}\right) + cn \\
&= bn(\log n - 1) + cn \\
&= bn\log n - (bn - cn) \\
&< bn\log n, \text{ if } b > c
\end{aligned}$$

Lower Bound?

- Show $T(n) \geq dn \log n$
- What can you assume?
  - $T(n) \geq 2T(n/2) + an$
  - $T(k) \geq dk \log k$, for $k < n$

- **Proof**

$$\begin{aligned}
T(n) &\geq 2T(n/2) + an \\
&\geq 2\left(d\frac{n}{2}\log\frac{n}{2}\right) + an \\
&= dn(\log n - 1) + an \\
&= dn\log n + (an - dn) \\
&\geq dn\log n, \text{ if } a \geq d
\end{aligned}$$

Master Theorem

- Consider a DAC algorithm with running time $T(n) = a\,T(n/b) + f(n)$
- Where $a \geq 1$ and $b > 1$ are constants and $f(n)$ positive; then:

1. If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
2. If $f(n) = \Theta(n^{\log_b a} \log^k n)$ with[1] $k \geq 0$, then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$
3. If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ with $\varepsilon > 0$, and $f(n)$ satisfies the regularity condition, then $T(n) = \Theta(f(n))$

- **Regularity condition:**

$af(n/b) \leq cf(n)$ for some $c < 1$ and large enough n

Master Theorem Examples

1. Matrix multiplication (Algorithm 1)

   $T(n) = 8T(n/2) + n^2 \in \Theta(n^3)$ Case 1

2. Matrix multiplication (Algorithm 1)

   $T(n) = 7T(n/2) + n^2 \in \Theta(n^{\log_2 7})$ Case 1

3. Max stock profit

   $T(n) = 2T(n/2) + n \in \Theta(n \log n)$ Case 2

4. Binary search

   $T(n) = T(n/2) + 1 \in \Theta(\log n)$ Case 2

5. A DAC algorithm with running time $T(n)$

   $T(n) = 2T(n/2) + n^2 \in \Theta(n^2)$ Case 3

END

Acknowledgements

- Introduction to Algorithms, 3rd edition, by T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein; MIT Press, 2009