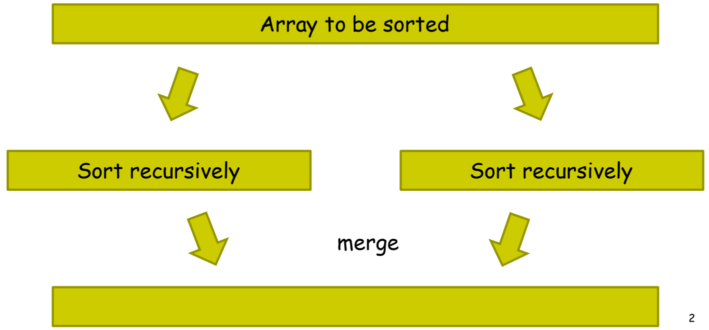
Algorithms

Lesson #1:
Divide and Conquer

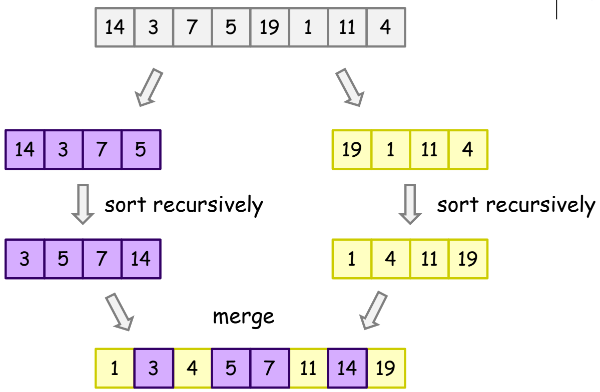
(Based on slides by Prof. Dana Shapira)

Divide and Conquer

Recall Merge Sort:



Merge Sort example:



Merge Sort runtime analysis

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + c'n & n > 1 \end{cases}$$

Solution: $T(n) = O(n \log n)$

(Evaluation by repeated substitution…)

The Master Theorem

3 examples:

$$f(n) = 4f\left(\frac{n}{3}\right) + n^{1.2}$$

Note that $\log_3 4 = 1.261 \ldots$

$$f(n) = 4f\left(\frac{n}{3}\right) + n^{1.3}$$

$$f(n) = 4f\left(\frac{n}{3}\right) + n^{\log_3 4}$$

(Evaluation by repeated substitution…)

The Master Theorem

Theorem: Given a recurrence relation of the form

$$f(n) = a \cdot f(n/b) + n^c$$

- If $c < \log_b a$ then $f(n) = \Theta(n^{\log_b a})$
- If $c = \log_b a$ then $f(n) = \Theta(n^{\log_b a} \log n)$
- If $c > \log_b a$ then $f(n) = \Theta(n^c)$

Another recurrence:

$$f(n) = f\left(\frac{3}{5}n\right) + f\left(\frac{n}{3}\right) + 7n$$

(Here repeated substitution gives a mess…)

Guess: $f(n) = O(n)$

Try to find a constant c such that we can prove by induction $f(n) \leq cn$

Divide and Conquer

Let ***A*** be an unsorted Array with ***n*** elements:

- Find the maximum element of ***A***
  - Exact number of comparisons – ***n***-1 (why?)
- Find the minimum element of ***A***
- Find maximum and minimum element of ***A***

MinMax(i,j)

```
MinMax(i,j){
if (j=i)
      return(A[i],A[i])
if (j=i+1) then
  if (A[i]<A[j]) then
      return(A[i],A[j])
  else return(A[j],A[i])
else {
  k ← ⌊(i+j)/2⌋
  (m1,M1)=MinMax(i,k)
  (m2,M2)=MinMax(k+1,j)
  return(min(m1,m2),max(M1,M2))
}
```

What is the number of comparisons?

$$T(n)=\begin{cases} 1 & n=2 \\ 2T\left(\frac{n}{2}\right)+2 & n>2 \end{cases}$$

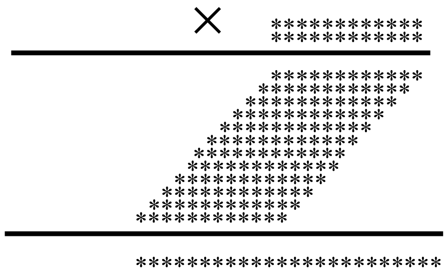
Boolean multiplication

X:

Y:

Question: How many bit operations needed for multiplying X and Y?

Boolean multiplication

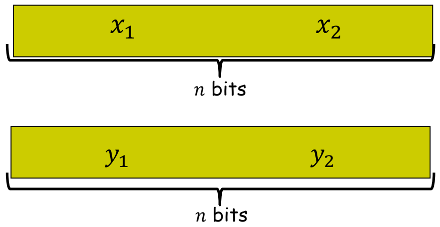


$\theta(n^2)$ bit operations.

Boolean multiplication

X:

Y:



$$x = x_2 + 2^{n/2} \cdot x_1$$

$$y = y_2 + 2^{n/2} \cdot y_1$$

Question: How many bit operations needed for multiplying X and Y?

Computing number of bit operations

$$x = x_2 + 2^{n/2} \cdot x_1$$

$$y = y_2 + 2^{n/2} \cdot y_1$$

$$x \cdot y = x_2 y_2 + 2^{n/2}\left(x_1 y_2 + x_2 y_1\right) + 2^n \cdot \left(x_1 y_1\right)$$

$$T(n) = \begin{cases} 1 & n = 1 \\ 4T\left(\frac{n}{2}\right) + cn & n > 1 \end{cases}$$

Question: What is the number of bit operations?
Is it worth it?

Improvement

$$A = x_1 y_1$$

$$B = x_2 y_2$$

$$C = \left(x_1 + x_2\right) \cdot \left(y_1 + y_2\right)$$

$$x \cdot y = x_2 y_2 + 2^{n/2}\left(x_1 y_2 + x_2 y_1\right) + 2^n \cdot \left(x_1 y_1\right) = B + 2^{n/2}\left(C - A - B\right) + 2^n \cdot A$$

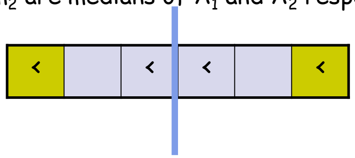
$$T(n) = \begin{cases} 1 & n = 1 \\ 3T\left(\frac{n}{2}\right) + c'n & n > 1 \end{cases}$$

Reminder - QuickSort

- Best Case
- Worst Case

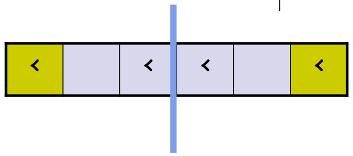
Median

- Problem: Given an unsorted array find its median
- Algorithms:
  1. Sort and return the n/2 element
  2. Divide and Conquer:
  - m_1 and m_2 are medians of A_1 and A_2 respectively

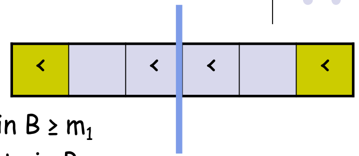


Median

```
Median(A){
   divide A into A1 and A2
   m1 ← Median(A1)
   m2 ← Median(A2)
   if (m1=m2)
        return m1
   if (m1<m2)
        Let B be the blue part of A
        m ← Median(B)
        return(m)
   else //    m1>m2
   …
   }
```



Median Algorithm Correctness



- **Claim 1:** $m_1 \leq m \leq m_2$
- Proof:
  - at least n/4 elements in B $\geq m_1$
  - $\Rightarrow$ At most n/4 elements in B < m_1
  - exactly n/4 elements in B < m
  - $\Rightarrow m_1 \leq m$
- **Claim 2:** m is the median of A
  - n/4 elements greater than m in B
  - n/4 elements greater than m_2 in A
  - $\Rightarrow$n/2 elements greater than m in A

Running time

```
Median(A){
   divide A into A1 and A2
   m1 ← Median(A1)
   m2 ← Median(A2)
   if (m1=m2)
        return m1
   if (m1<m2)
        Let B be the blue part of A
        m ← Median(B)
        return(m)
   else //    m1>m2
   …
   }
```

What is the running time?

$$T(n) = \begin{cases} 1 & n = 1 \\ 3T\left(\frac{n}{2}\right) + n & n > 1 \end{cases}$$

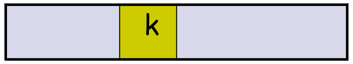
Corollary: Solution 1 is better…

Median

- Algorithms:
  1. Sort and return the $n/2$ element
  2. Divide and Conquer
  3. Define: SELECT(A,t) returns the t-th element in A.
  - The median is SELECT(A,$n/2$)

SELECT(A,t)

```
SELECT(A,t){
  choose a pivot k randomly
  Let:
```



$S_1 = \{x \in A | x < k\}$

$S_2 = \{x \in A | x > k\}$

```
  if |S_1|=t-1
      return k
  else if |S_1|>t-1
      SELECT(S_1,t)
  else // |S_1|<t-1
      SELECT(S_2,t-|S_1|-1)
  }
```

What is the running time in the worst case?

Controlling the pivot

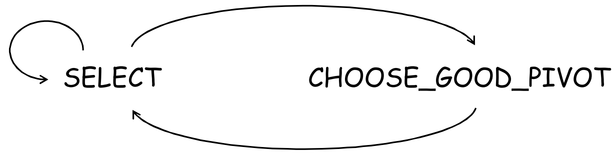
```
SELECT(A,t){
if |A|<50 sort A and return the t-th element
k ← CHOOSE_GOOD_PIVOT(A)
let: S_1 = {x ∈ A|x < k}
     S_2 = {x ∈ A|x > k}
if |S_1|=t-1
   return k
else if |S_1|>t-1
   SELECT(S_1,t)
else // |S_1|<t-1
   SELECT(S_2,t-|S_1|-1)
}
```

doesn't matter so much

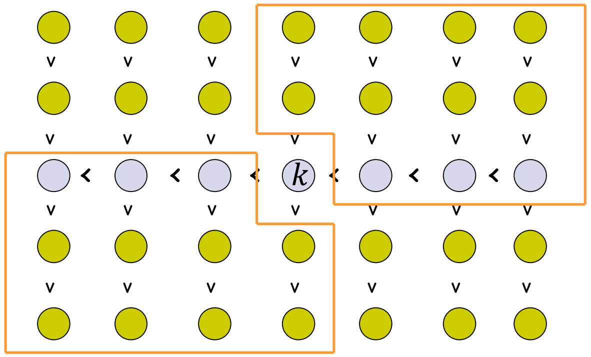
Controlling the pivot

```
CHOOSE_GOOD_PIVOT(A){
divide A into groups of size 5.
Sort each group
B={medians of the groups}
k ← SELECT(B,n/10)
}
```

"Median-of-medians algorithm"



Running Time?



Running Time?

- $n/5$ groups
- $n/10$ groups with medians $< k$
- Each such group has 3 elements $<$ median of group $\Rightarrow < k$
- $\Rightarrow$ At least $3n/10$ elements $< k$
- $\Rightarrow$ At least $3n/10$ elements $> k$
- $T(n) \leq \begin{cases} c' & n < 50 \\ cn + T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) & n \geq 50 \end{cases}$

Running Time?

- $T(n) \leq \begin{cases} c' & n < 50 \\ cn + T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) & n \geq 50 \end{cases}$

- **Claim:** SELECT runs in linear time
  - **Proof:** There exists a constant d such that $T(n) \leq d \cdot n$.
  - Divide to $n < 50$ and $n \geq 50$

Now we can “fix” QuickSort…

…so that it runs in time $O(n \log n)$ in the worst case:

- Use SELECT to take the median as the pivot.