CS 170: Algorithms

CS 170: Algorithms

Hello and ...

CS 170: Algorithms

Hello and ...

S

CS 170: Algorithms

Hello and ...

s $\textcolor{blue}{H}$

CS 170: Algorithms

Hello and ...

S H H

CS 170: Algorithms

Hello and ...

S H H H

CS 170: Algorithms

Hello and ...

S H H H H

CS 170: Algorithms

Hello and ...

S H H H H .

CS 170: Algorithms

Hello and ...

S H H H H . .

CS 170: Algorithms

Hello and ...

S H H H H . . .

CS 170: Algorithms

Hello and ...

S H H H H . . . .

CS 170: Algorithms

Hello and ...

S H H H H . . . . .

CS 170: Algorithms

Hello and ...

S H H H H . . . . .

More divide and conquer: mergesort.

Sort items in n elt array: $A = [a_1, \ldots, a_n]$,

More divide and conquer: mergesort.

Sort items in n elt array: $A = [a_1, \ldots, a_n]$,
E.g., $A = [5, 6, 7, 9, 10, 2, 3...]$.

More divide and conquer: mergesort.

Sort items in n elt array: $A = [a_1, \ldots, a_n]$,
E.g., $A = [5,6,7,9,10,2,3...]$.

Mergesort(A)

```
if (length(A) >1)
   return
      (merge(mergesort(a[1],...,a[n/2]),
             mergesort(a[n/2+1],...,a[n]))
else
   return a
```

More divide and conquer: mergesort.

Sort items in n elt array: $A = [a_1, \ldots, a_n]$,
E.g., $A = [5, 6, 7, 9, 10, 2, 3...]$.

Mergesort(A)

```
if (length(A) >1)
   return
      (merge(mergesort(a[1],...,a[n/2]),
             mergesort(a[n/2+1],...,a[n]))
else
   return a
```

How to merge?

More divide and conquer: mergesort.

Sort items in n elt array: $A = [a_1, \ldots, a_n]$,
E.g., $A = [5, 6, 7, 9, 10, 2, 3...]$.

Mergesort(A)

```
if (length(A) >1)
   return
      (merge(mergesort(a[1],...,a[n/2]),
             mergesort(a[n/2+1],...,a[n]))
else
   return a
```

How to merge?

Choose lowest from two lists,

More divide and conquer: mergesort.

Sort items in n elt array: $A = [a_1, \ldots, a_n]$,
E.g., $A = [5, 6, 7, 9, 10, 2, 3 ...]$.

Mergesort(A)

```
if (length(A) >1)
   return
      (merge(mergesort(a[1],...,a[n/2]),
             mergesort(a[n/2+1],...,a[n]))
else
   return a
```

How to merge?

Choose lowest from two lists, cross out,

More divide and conquer: mergesort.

Sort items in n elt array: $A = [a_1, \ldots, a_n]$,
E.g., $A = [5, 6, 7, 9, 10, 2, 3...]$.

Mergesort(A)

```
if (length(A) >1)
   return
      (merge(mergesort(a[1],...,a[n/2]),
             mergesort(a[n/2+1],...,a[n]))
else
   return a
```

How to merge?

Choose lowest from two lists, cross out, repeat.

More divide and conquer: mergesort.

Sort items in n elt array: $A = [a_1, \ldots, a_n]$,
E.g., $A = [5, 6, 7, 9, 10, 2, 3...]$.

Mergesort(A)

```
if (length(A) >1)
    return
        (merge(mergesort(a[1],...,a[n/2]),
               mergesort(a[n/2+1],...,a[n]))
else
    return a
```

How to merge?

Choose lowest from two lists, cross out, repeat.

Sorted SubArray 1: $3\,, 7\,, 8\,, 10, 11, \ldots$
Sorted Subarray 2: $4\,, 5\,, 9, 19, 20, \ldots$

$, \;\; , \;\; , \;\; ,$

More divide and conquer: mergesort.

Sort items in n elt array: $A = [a_1, \ldots, a_n]$,
E.g., $A = [5, 6, 7, 9, 10, 2, 3...]$.

Mergesort(A)

```
if (length(A) >1)
   return
      (merge(mergesort(a[1],...,a[n/2]),
             mergesort(a[n/2+1],...,a[n]))
else
   return a
```

How to merge?

Choose lowest from two lists, cross out, repeat.

Sorted SubArray 1: $\cancel{3}, 7, 8, 10, 11, \ldots$

Sorted Subarray 2: $4, 5, 9, 19, 20, \ldots$

3, , , ,

More divide and conquer: mergesort.

Sort items in n elt array: $A = [a_1, \ldots, a_n]$,
E.g., $A = [5, 6, 7, 9, 10, 2, 3...]$.

Mergesort(A)

```
if (length(A) >1)
   return
      (merge(mergesort(a[1],...,a[n/2]),
            mergesort(a[n/2+1],...,a[n]))
else
   return a
```

How to merge?

Choose lowest from two lists, cross out, repeat.

Sorted SubArray 1: $\cancel{3}, 7, 8, 10, 11, \ldots$
Sorted Subarray 2: $\cancel{4}, 5, 9, 19, 20, \ldots$

$3, 4, \;, \;,$

More divide and conquer: mergesort.

Sort items in n elt array: $A = [a_1, \ldots, a_n]$,
E.g., $A = [5, 6, 7, 9, 10, 2, 3...]$.

Mergesort(A)

```
if (length(A) >1)
   return
      (merge(mergesort(a[1],...,a[n/2]),
             mergesort(a[n/2+1],...,a[n]))
else
   return a
```

How to merge?

Choose lowest from two lists, cross out, repeat.

Sorted SubArray 1: $\cancel{3}, 7, 8, 10, 11, \ldots$

Sorted Subarray 2: $\cancel{4}, \cancel{5}, 9, 19, 20, \ldots$

$3, 4, 5, \ ,$

More divide and conquer: mergesort.

Sort items in n elt array: $A = [a_1, \ldots, a_n]$,
E.g., $A = [5, 6, 7, 9, 10, 2, 3...]$.

Mergesort(A)

```
if (length(A) >1)
    return
        (merge(mergesort(a[1],...,a[n/2]),
               mergesort(a[n/2+1],...,a[n]))
else
    return a
```

How to merge?

Choose lowest from two lists, cross out, repeat.

Sorted SubArray 1: $\cancel{3}, \cancel{7}, 8, 10, 11, \ldots$

Sorted Subarray 2: $\cancel{4}, \cancel{5}, 9, 19, 20, \ldots$

$3, 4, 5, 7,$

More divide and conquer: mergesort.

Sort items in n elt array: $A = [a_1, \ldots, a_n]$,
E.g., $A = [5, 6, 7, 9, 10, 2, 3 ...]$.

Mergesort(A)

```
if (length(A) >1)
   return
      (merge(mergesort(a[1],...,a[n/2]),
             mergesort(a[n/2+1],...,a[n]))
else
   return a
```

How to merge?

Choose lowest from two lists, cross out, repeat.

Sorted SubArray 1: $\cancel{3}, \cancel{7}, \cancel{8}, 10, 11, \ldots$

Sorted Subarray 2: $\cancel{4}, \cancel{5}, 9, 19, 20, \ldots$

$3, 4, 5, 7, 8$

More divide and conquer: mergesort.

Sort items in n elt array: $A = [a_1, \ldots, a_n]$,
E.g., $A = [5,6,7,9,10,2,3...]$.

Mergesort(A)

```
if (length(A) >1)
   return
      (merge(mergesort(a[1],...,a[n/2]),
             mergesort(a[n/2+1],...,a[n]))
else
   return a
```

How to merge?

Choose lowest from two lists, cross out, repeat.

Sorted SubArray 1: $\cancel{3}, \cancel{7}, \cancel{8}, 10, 11, \ldots$

Sorted Subarray 2: $\cancel{4}, \cancel{5}, 9, 19, 20, \ldots$

$3,4,5,7,8 \quad \ldots$

Mergesort: running time analysis

Mergesort(A)

```
if (length(A) >1)
   return
      (merge(mergesort( a[1],...,a[n/2]),
             mergesort( a[n/2+1],...,a[n]))
else
   return a
```

Mergesort: running time analysis

Mergesort(A)

```
if (length(A) >1)
   return
      (merge(mergesort( a[1],...,a[n/2]),
             mergesort( a[n/2+1],...,a[n]))
else
   return a
```

Split: $O(n)$ time

Mergesort: running time analysis

Mergesort(A)

```
if (length(A) >1)
    return
        (merge(mergesort( a[1],...,a[n/2]),
               mergesort( a[n/2+1],...,a[n]))
else
    return a
```

Split: $O(n)$ time
Could be $O(1)$, e.g., **MergeSort(A,start,finish)**.

Mergesort: running time analysis

Mergesort(A)

```
if (length(A) >1)
    return
        (merge(mergesort( a[1],...,a[n/2]),
               mergesort( a[n/2+1],...,a[n]))
else
    return a
```

Split: $O(n)$ time

Could be $O(1)$, e.g., **MergeSort(A,start,finish)**.

Merge:

Mergesort: running time analysis

Mergesort(A)

```
if (length(A) >1)
    return
        (merge(mergesort( a[1],...,a[n/2]),
                mergesort( a[n/2+1],...,a[n]))
else
    return a
```

Split: $O(n)$ time

Could be $O(1)$, e.g., **MergeSort(A,start,finish)**.

Merge: each element in output takes one comparision

Mergesort: running time analysis

Mergesort(A)

```
if (length(A) >1)
    return
        (merge(mergesort( a[1],...,a[n/2]),
               mergesort( a[n/2+1],...,a[n]))
else
    return a
```

Split: $O(n)$ time

Could be $O(1)$, e.g., **MergeSort(A,start,finish)**.

Merge: each element in output takes one comparision : $O(n)$.

Mergesort: running time analysis

Mergesort(A)

```
if (length(A) >1)
    return
        (merge(mergesort( a[1],...,a[n/2]),
               mergesort( a[n/2+1],...,a[n]))
else
    return a
```

Split: $O(n)$ time

Could be $O(1)$, e.g., **MergeSort(A,start,finish)**.

Merge: each element in output takes one comparision : $O(n)$.

Recursive: 2 subproblems of size $n/2$.

Mergesort: running time analysis

Mergesort(A)

```
if (length(A) >1)
   return
      (merge(mergesort( a[1],...,a[n/2]),
             mergesort( a[n/2+1],...,a[n]))
else
   return a
```

Split: $O(n)$ time

Could be $O(1)$, e.g., **MergeSort(A,start,finish)**.

Merge: each element in output takes one comparision : $O(n)$.

Recursive: 2 subproblems of size $n/2$.

$$T(n) = 2T(\frac{n}{2}) + O(n).$$

Mergesort: running time analysis

Mergesort(A)

```
if (length(A) >1)
    return
        (merge(mergesort( a[1],...,a[n/2]),
               mergesort( a[n/2+1],...,a[n]))
else
    return a
```

Split: $O(n)$ time
Could be $O(1)$, e.g., **MergeSort(A,start,finish)**.
Merge: each element in output takes one comparision : $O(n)$.
Recursive: 2 subproblems of size $n/2$.

$$T(n) = 2T(\frac{n}{2}) + O(n).$$

Masters: $T(n) = aT(n/b) + O(n^d)$

Mergesort: running time analysis

Mergesort(A)

```
if (length(A) >1)
    return
        (merge(mergesort( a[1],...,a[n/2]),
               mergesort( a[n/2+1],...,a[n]))
else
    return a
```

Split: $O(n)$ time
Could be $O(1)$, e.g., **MergeSort(A,start,finish)**.
Merge: each element in output takes one comparision : $O(n)$.
Recursive: 2 subproblems of size $n/2$.

$$T(n) = 2T(\frac{n}{2}) + O(n).$$

Masters: $T(n) = aT(n/b) + O(n^d)$
$a = 2$, $b = 2$, $d = 1$

Mergesort: running time analysis

Mergesort(A)

```
if (length(A) >1)
   return
      (merge(mergesort( a[1],...,a[n/2]),
             mergesort( a[n/2+1],...,a[n]))
else
   return a
```

Split: $O(n)$ time
 Could be $O(1)$, e.g., **MergeSort(A,start,finish)**.
Merge: each element in output takes one comparision : $O(n)$.
Recursive: 2 subproblems of size $n/2$.

$$T(n) = 2T(\frac{n}{2}) + O(n).$$

Masters: $T(n) = aT(n/b) + O(n^d)$
$a = 2$, $b = 2$, $d = 1$
$\implies \log_2 2 = 1$

Mergesort: running time analysis

Mergesort(A)

```
  if (length(A) >1)
     return
        (merge(mergesort( a[1],...,a[n/2]),
               mergesort( a[n/2+1],...,a[n]))
  else
     return a
```

Split: $O(n)$ time
 Could be $O(1)$, e.g., **MergeSort(A,start,finish)**.
Merge: each element in output takes one comparision : $O(n)$.
Recursive: 2 subproblems of size $n/2$.

$$T(n) = 2T(\frac{n}{2}) + O(n).$$

Masters: $T(n) = aT(n/b) + O(n^d)$
$a = 2$, $b = 2$, $d = 1$
$\implies \log_2 2 = 1$
Masters: $d == \log_b a$

Mergesort: running time analysis

Mergesort(A)

```
if (length(A) >1)
   return
      (merge(mergesort( a[1],...,a[n/2]),
             mergesort( a[n/2+1],...,a[n]))
else
   return a
```

Split: $O(n)$ time

Could be $O(1)$, e.g., **MergeSort(A,start,finish)**.

Merge: each element in output takes one comparision : $O(n)$.

Recursive: 2 subproblems of size $n/2$.

$$T(n) = 2T(\frac{n}{2}) + O(n).$$

Masters: $T(n) = aT(n/b) + O(n^d)$

$a = 2$, $b = 2$, $d = 1$

$\implies \log_2 2 = 1$

Masters: $d == \log_b a \implies O(n^d \log_b n)$

Mergesort: running time analysis

Mergesort(A)

```
if (length(A) >1)
    return
        (merge(mergesort( a[1],...,a[n/2]),
               mergesort( a[n/2+1],...,a[n]))
else
    return a
```

Split: $O(n)$ time

Could be $O(1)$, e.g., **MergeSort(A,start,finish)**.

Merge: each element in output takes one comparision : $O(n)$.

Recursive: 2 subproblems of size $n/2$.

$$T(n) = 2T(\frac{n}{2}) + O(n).$$

Masters: $T(n) = aT(n/b) + O(n^d)$

$a = 2$, $b = 2$, $d = 1$

$\implies \log_2 2 = 1$

Masters: $d == \log_b a \implies O(n^d \log_b n) \implies T(n) = O(n \log n)$.

Check it out...

Iterative Mergesort: Bottom up, use of queues.

Check it out...

Iterative Mergesort: Bottom up, use of queues.

Make each element into list and put lists in queue.

Check it out...

Iterative Mergesort: Bottom up, use of queues.

Make each element into list and put lists in queue.

$\boxed{8}$ $\boxed{3}$ $\boxed{5}$ $\boxed{9}$ $\cdots\cdots$

Check it out...

Iterative Mergesort: Bottom up, use of queues.

Make each element into list and put lists in queue.

$\boxed{8}$ $\boxed{3}$ $\boxed{5}$ $\boxed{9}$ $\cdots\cdots$

Merge first two lists, put in queue (at end).

$\boxed{5}$ $\boxed{9}$ $\cdots\cdots$ $\boxed{3,8}$

Check it out...

Iterative Mergesort: Bottom up, use of queues.

Make each element into list and put lists in queue.

$\boxed{8}$ $\boxed{3}$ $\boxed{5}$ $\boxed{9}$ $\cdots\cdots\cdot$

Merge first two lists, put in queue (at end).

$\boxed{5}$ $\boxed{9}$ $\cdots\cdots\cdot$ $\boxed{3,8}$

Rinse.

Check it out...

Iterative Mergesort: Bottom up, use of queues.

Make each element into list and put lists in queue.

$\boxed{8}$ $\boxed{3}$ $\boxed{5}$ $\boxed{9}$ $\cdots\cdots\cdot$

Merge first two lists, put in queue (at end).

$\boxed{5}$ $\boxed{9}$ $\cdots\cdots\cdot$ $\boxed{3,8}$

Rinse. Repeat.

Check it out...

Iterative Mergesort: Bottom up, use of queues.

Make each element into list and put lists in queue.

$\boxed{8}$ $\boxed{3}$ $\boxed{5}$ $\boxed{9}$ $\cdots\cdots\cdot$

Merge first two lists, put in queue (at end).

$\boxed{5}$ $\boxed{9}$ $\cdots\cdots\cdot$ $\boxed{3,8}$

Rinse. Repeat.

$\cdots\cdots\cdot$ $\boxed{3,8}$ $\boxed{5,9}$

Check it out...

Iterative Mergesort: Bottom up, use of queues.

Make each element into list and put lists in queue.

$\boxed{\boxed{8}\ \boxed{3}\ \boxed{5}\ \boxed{9}\ \cdots\cdots\cdot}$

Merge first two lists, put in queue (at end).

$\boxed{\boxed{5}\ \boxed{9}\ \cdots\cdots\cdot\ \boxed{3,8}}$

Rinse. Repeat.

$\boxed{\cdots\cdots\cdot\ \boxed{3,8}\ \boxed{5,9}}$

And next pass through queue...

Check it out...

Iterative Mergesort: Bottom up, use of queues.

Make each element into list and put lists in queue.

$$\boxed{8} \quad \boxed{3} \quad \boxed{5} \quad \boxed{9} \qquad \cdots\cdots\cdot$$

Merge first two lists, put in queue (at end).

$$\boxed{5} \quad \boxed{9} \qquad \cdots\cdots\cdot \qquad \boxed{3,8}$$

Rinse. Repeat.

$$\cdots\cdots\cdot \qquad \boxed{3,8} \qquad \boxed{5,9}$$

And next pass through queue...

$$\cdots\cdots\cdot \qquad \boxed{3,5,8,9}$$

Check it out...

Iterative Mergesort: Bottom up, use of queues.

Make each element into list and put lists in queue.

$\boxed{8}\ \boxed{3}\ \boxed{5}\ \boxed{9}\ \cdots\cdots$

Merge first two lists, put in queue (at end).

$\boxed{5}\ \boxed{9}\ \cdots\cdots\ \boxed{3,8}$

Rinse. Repeat.

$\cdots\cdots\ \boxed{3,8}\ \boxed{5,9}$

And next pass through queue...

$\cdots\cdots\ \boxed{3,5,8,9}$

Each pass through queue:

Check it out...

Iterative Mergesort: Bottom up, use of queues.

Make each element into list and put lists in queue.

$\boxed{8}\ \boxed{3}\ \boxed{5}\ \boxed{9}\ \cdots\cdots$

Merge first two lists, put in queue (at end).

$\boxed{5}\ \boxed{9}\ \cdots\cdots\ \boxed{3,8}$

Rinse. Repeat.

$\cdots\cdots\ \boxed{3,8}\ \boxed{5,9}$

And next pass through queue...

$\cdots\cdots\ \boxed{3,5,8,9}$

Each pass through queue: each element touched once.

Check it out...

Iterative Mergesort: Bottom up, use of queues.

Make each element into list and put lists in queue.

$\boxed{8}$ $\boxed{3}$ $\boxed{5}$ $\boxed{9}$ $\cdots\cdots$

Merge first two lists, put in queue (at end).

$\boxed{5}$ $\boxed{9}$ $\cdots\cdots$ $\boxed{3,8}$

Rinse. Repeat.

$\cdots\cdots$ $\boxed{3,8}$ $\boxed{5,9}$

And next pass through queue...

$\cdots\cdots$ $\boxed{3,5,8,9}$

Each pass through queue: each element touched once. $O(n)$ time.

Check it out...

Iterative Mergesort: Bottom up, use of queues.

Make each element into list and put lists in queue.

$\boxed{8}\ \boxed{3}\ \boxed{5}\ \boxed{9}\ \cdots\cdots\cdot$

Merge first two lists, put in queue (at end).

$\boxed{5}\ \boxed{9}\ \cdots\cdots\cdot\ \boxed{3,8}$

Rinse. Repeat.

$\cdots\cdots\cdot\ \boxed{3,8}\ \boxed{5,9}$

And next pass through queue...

$\cdots\cdots\cdot\ \boxed{3,5,8,9}$

Each pass through queue: each element touched once. $O(n)$ time.

Each pass halves number of lists.

Check it out...

Iterative Mergesort: Bottom up, use of queues.

Make each element into list and put lists in queue.

$\boxed{8}\ \boxed{3}\ \boxed{5}\ \boxed{9}\ \cdots\cdots\cdot$

Merge first two lists, put in queue (at end).

$\boxed{5}\ \boxed{9}\ \cdots\cdots\cdot\ \boxed{3,8}$

Rinse. Repeat.

$\cdots\cdots\cdot\ \boxed{3,8}\ \boxed{5,9}$

And next pass through queue...

$\cdots\cdots\cdot\ \boxed{3,5,8,9}$

Each pass through queue: each element touched once. $O(n)$ time.

Each pass halves number of lists. $\implies$ $O(\log n)$ passes

Check it out...

Iterative Mergesort: Bottom up, use of queues.

Make each element into list and put lists in queue.

$\boxed{8}$ $\boxed{3}$ $\boxed{5}$ $\boxed{9}$ $\cdots\cdots\cdot$

Merge first two lists, put in queue (at end).

$\boxed{5}$ $\boxed{9}$ $\cdots\cdots\cdot$ $\boxed{3,8}$

Rinse. Repeat.

$\cdots\cdots\cdot$ $\boxed{3,8}$ $\boxed{5,9}$

And next pass through queue...

$\cdots\cdots\cdot$ $\boxed{3,5,8,9}$

Each pass through queue: each element touched once. $O(n)$ time.
Each pass halves number of lists. $\implies$ $O(\log n)$ passes $\implies$ $O(n \log n)$ time

Sorting lower bound.

Can we do better?

Sorting lower bound.

Can we do better?

Comparison sorting algorithm only compares numbers.

Sorting lower bound.

Can we do better?

Comparison sorting algorithm only compares numbers.
Does not look at bits.

Sorting lower bound.

Can we do better?

Comparison sorting algorithm only compares numbers.
Does not look at bits.

Merge:
Compare two first elts and then output first.

Sorting lower bound.

Can we do better?

Comparison sorting algorithm only compares numbers.
Does not look at bits.

Merge:
Compare two first elts and then output first.

Comparison sort?

Sorting lower bound.

Can we do better?

Comparison sorting algorithm only compares numbers.
Does not look at bits.

Merge:
Compare two first elts and then output first.

Comparison sort? Yes.

Sorting lower bound.

Can we do better?

Comparison sorting algorithm only compares numbers.
Does not look at bits.

Merge:
Compare two first elts and then output first.

Comparison sort? Yes.

“Radix” Sort.

Sorting lower bound.

Can we do better?

Comparison sorting algorithm only compares numbers.
Does not look at bits.

Merge:
Compare two first elts and then output first.

Comparison sort? Yes.

“Radix” Sort.
Bucket according to whether begins with “A”, “B”....

Sorting lower bound.

Can we do better?

Comparison sorting algorithm only compares numbers.
Does not look at bits.

Merge:
Compare two first elts and then output first.

Comparison sort? Yes.

“Radix” Sort.
Bucket according to whether begins with “A”, “B”....
Repeat in each bucket with next characters.

Sorting lower bound.

Can we do better?

Comparison sorting algorithm only compares numbers.
Does not look at bits.

Merge:
Compare two first elts and then output first.

Comparison sort? Yes.

"Radix" Sort.
Bucket according to whether begins with "A", "B"....
Repeat in each bucket with next characters.
Looks at characters...

Sorting lower bound.

Can we do better?

Comparison sorting algorithm only compares numbers.
Does not look at bits.

Merge:
Compare two first elts and then output first.

Comparison sort? Yes.

“Radix” Sort.
Bucket according to whether begins with “A”, “B”....
Repeat in each bucket with next characters.
Looks at characters... or looks at “bits”.

Sorting lower bound.

Can we do better?

Comparison sorting algorithm only compares numbers.
Does not look at bits.

Merge:
Compare two first elts and then output first.

Comparison sort? Yes.

“Radix” Sort.
Bucket according to whether begins with “A”, “B”....
Repeat in each bucket with next characters.
Looks at characters... or looks at “bits”.
Not a comparision sort.!

Sorting lower bound.

Thm: Comparison sort requires $\Omega(n \log n)$ comparisons.

Sorting lower bound.

Thm: Comparison sort requires $\Omega(n \log n)$ comparisons.

Proof idea: Input: $a_1, a_2, \ldots, a_n$

Sorting lower bound.

Thm: Comparison sort requires $\Omega(n \log n)$ comparisons.

Proof idea: Input: $a_1, a_2, \ldots, a_n$
Possible Output: $a_8, a_{n-8}, \ldots, a_{15}$

Sorting lower bound.

Thm: Comparison sort requires $\Omega(n \log n)$ comparisons.

Proof idea: Input: $a_1, a_2, \ldots, a_n$
Possible Output: $a_8, a_{n-8}, \ldots, a_{15}$
Represent output as permuation of $[1, \ldots, n]$.

Sorting lower bound.

Thm: Comparison sort requires $\Omega(n \log n)$ comparisons.

Proof idea: Input: $a_1, a_2, \ldots, a_n$
Possible Output: $a_8, a_{n-8}, \ldots, a_{15}$
Represent output as permuation of $[1, \ldots, n]$.
Output: $8, n-8, \ldots, 15$.

Sorting lower bound.

Thm: Comparison sort requires $\Omega(n \log n)$ comparisons.

Proof idea: Input: $a_1, a_2, \ldots, a_n$
Possible Output: $a_8, a_{n-8}, \ldots, a_{15}$
Represent output as permuation of $[1, \ldots, n]$.
Output: $8, n-8, \ldots, 15$.

How many possible outputs?

Sorting lower bound.

Thm: Comparison sort requires $\Omega(n \log n)$ comparisons.

Proof idea: Input: $a_1, a_2, \ldots, a_n$
Possible Output: $a_8, a_{n-8}, \ldots, a_{15}$
Represent output as permuation of $[1, \ldots, n]$.
Output: $8, n-8, \ldots, 15$.

How many possible outputs? $n!$

Sorting lower bound.

Thm: Comparison sort requires $\Omega(n \log n)$ comparisons.

Proof idea: Input: $a_1, a_2, \ldots, a_n$
Possible Output: $a_8, a_{n-8}, \ldots, a_{15}$
Represent output as permuation of $[1, \ldots, n]$.
Output: $8, n-8, \ldots, 15$.

How many possible outputs? $n!$

Algorithm could output any of $n!$ permuations.

Sorting lower bound.

Thm: Comparison sort requires $\Omega(n \log n)$ comparisons.

Proof idea: Input: $a_1, a_2, \ldots, a_n$
Possible Output: $a_8, a_{n-8}, \ldots, a_{15}$
Represent output as permuation of $[1, \ldots, n]$.
Output: $8, n-8, \ldots, 15$.

How many possible outputs? $n!$

Algorithm could output any of $n!$ permuations.

Algorithm must output just 1 permutation at termination.

Sorting lower bound: ...proof

Algorithm must be able to output any of n! permuations.

Sorting lower bound: ...proof

Algorithm must be able to output any of n! permuations.
Algorithm must output just 1 permutation at termination.

Sorting lower bound: ...proof

Algorithm must be able to output any of n! permuations.
Algorithm must output just 1 permutation at termination.
S is set of possible permutations at some point in Algorithm

Sorting lower bound: ...proof

Algorithm must be able to output any of n! permuations.
Algorithm must output just 1 permutation at termination.
S is set of possible permutations at some point in Algorithm

Do some comparision: $a_i > a_j$?

Sorting lower bound: ...proof

Algorithm must be able to output any of n! permuations.
Algorithm must output just 1 permutation at termination.
S is set of possible permutations at some point in Algorithm

Do some comparision: $a_i > a_j$?

Sorting lower bound: ...proof

Algorithm must be able to output any of n! permuations.
Algorithm must output just 1 permutation at termination.
S is set of possible permutations at some point in Algorithm

Do some comparision: $a_i > a_j$?

If Yes, Alg "could" return subset of permutations: S_1.

Sorting lower bound: ...proof

Algorithm must be able to output any of n! permuations.
Algorithm must output just 1 permutation at termination.
S is set of possible permutations at some point in Algorithm

Do some comparision: $a_i > a_j$?

If Yes, Alg “could” return subset of permutations: S_1.
If No, Alg “could” return subset of permutations: S_2.

Sorting lower bound: ...proof

Algorithm must be able to output any of $n!$ permuations.
Algorithm must output just 1 permutation at termination.
S is set of possible permutations at some point in Algorithm

Do some comparision: $a_i > a_j$?

If Yes, Alg "could" return subset of permutations: S_1.
If No, Alg "could" return subset of permutations: S_2.

$S_1 \cup S_2$ has to be S.

Sorting lower bound: ...proof

Algorithm must be able to output any of n! permuations.
Algorithm must output just 1 permutation at termination.
S is set of possible permutations at some point in Algorithm

Do some comparision: $a_i > a_j$?

If Yes, Alg "could" return subset of permutations: S_1.
If No, Alg "could" return subset of permutations: S_2.

$S_1 \cup S_2$ has to be S.

One of S_1 or S_2 contains at least half of S.

Sorting lower bound: ...proof

Algorithm must be able to output any of n! permuations.
Algorithm must output just 1 permutation at termination.
S is set of possible permutations at some point in Algorithm

Do some comparision: $a_i > a_j$?

If Yes, Alg "could" return subset of permutations: S_1.
If No, Alg "could" return subset of permutations: S_2.

$S_1 \cup S_2$ has to be S.

One of S_1 or S_2 contains at least half of S.

Consider larger of S_1 or S_2 and next comparision ...

Sorting lower bound: ...proof

Algorithm must be able to output any of $n!$ permuations.
Algorithm must output just 1 permutation at termination.
S is set of possible permutations at some point in Algorithm

Do some comparision: $a_i > a_j$?

If Yes, Alg "could" return subset of permutations: S_1.
If No, Alg "could" return subset of permutations: S_2.

$S_1 \cup S_2$ has to be S.

One of S_1 or S_2 contains at least half of S.

Consider larger of S_1 or S_2 and next comparision ...
... divides size of output set by at most 2.

Sorting lower bound: ...proof

Algorithm must be able to output any of $n!$ permuations.
Algorithm must output just 1 permutation at termination.
S is set of possible permutations at some point in Algorithm

Do some comparision: $a_i > a_j$?

If Yes, Alg "could" return subset of permutations: S_1.
If No, Alg "could" return subset of permutations: S_2.

$S_1 \cup S_2$ has to be S.

One of S_1 or S_2 contains at least half of S.

Consider larger of S_1 or S_2 and next comparision …
… divides size of output set by at most 2.

Each comparision divides possible outputs by at most 2.

Sorting lower bound: ...proof

Algorithm must be able to output any of $n!$ permuations.
Algorithm must output just 1 permutation at termination.
S is set of possible permutations at some point in Algorithm

Do some comparision: $a_i > a_j$?

If Yes, Alg "could" return subset of permutations: S_1.
If No, Alg "could" return subset of permutations: S_2.

$S_1 \cup S_2$ has to be S.

One of S_1 or S_2 contains at least half of S.

Consider larger of S_1 or S_2 and next comparision ...
... divides size of output set by at most 2.

Each comparision divides possible outputs by at most 2.

Need at least $\log_2(n!)$ comparisions to get to just 1 permutation.

Sorting lower bound: ...proof

Algorithm must be able to output any of $n!$ permuations.
Algorithm must output just 1 permutation at termination.
S is set of possible permutations at some point in Algorithm

Do some comparision: $a_i > a_j$?

If Yes, Alg "could" return subset of permutations: S_1.
If No, Alg "could" return subset of permutations: S_2.

$S_1 \cup S_2$ has to be S.

One of S_1 or S_2 contains at least half of S.

Consider larger of S_1 or S_2 and next comparision ...
... divides size of output set by at most 2.

Each comparision divides possible outputs by at most 2.

Need at least $\log_2(n!)$ comparisions to get to just 1 permutation.
...to get to termination.

Sorting lower bound: ...proof

Algorithm must be able to output any of $n!$ permuations.
Algorithm must output just 1 permutation at termination.
S is set of possible permutations at some point in Algorithm

Do some comparision: $a_i > a_j$?

If Yes, Alg "could" return subset of permutations: S_1.
If No, Alg "could" return subset of permutations: S_2.

$S_1 \cup S_2$ has to be S.

One of S_1 or S_2 contains at least half of S.

Consider larger of S_1 or S_2 and next comparision ...
... divides size of output set by at most 2.

Each comparision divides possible outputs by at most 2.

Need at least $\log_2(n!)$ comparisions to get to just 1 permutation.
...to get to termination.

$n! \geq (\frac{n}{e})^n$

Sorting lower bound: ...proof

Algorithm must be able to output any of $n!$ permuations.
Algorithm must output just 1 permutation at termination.
S is set of possible permutations at some point in Algorithm

Do some comparision: $a_i > a_j$?

If Yes, Alg "could" return subset of permutations: S_1.
If No, Alg "could" return subset of permutations: S_2.

$S_1 \cup S_2$ has to be S.

One of S_1 or S_2 contains at least half of S.

Consider larger of S_1 or S_2 and next comparision ...
... divides size of output set by at most 2.

Each comparision divides possible outputs by at most 2.

Need at least $\log_2(n!)$ comparisions to get to just 1 permutation.
...to get to termination.

$n! \geq (\frac{n}{e})^n \implies \log n!$

Sorting lower bound: ...proof

Algorithm must be able to output any of $n!$ permuations.
Algorithm must output just 1 permutation at termination.
S is set of possible permutations at some point in Algorithm

Do some comparision: $a_i > a_j$?

If Yes, Alg "could" return subset of permutations: S_1.
If No, Alg "could" return subset of permutations: S_2.

$S_1 \cup S_2$ has to be S.

One of S_1 or S_2 contains at least half of S.

Consider larger of S_1 or S_2 and next comparision ...
... divides size of output set by at most 2.

Each comparision divides possible outputs by at most 2.

Need at least $\log_2(n!)$ comparisions to get to just 1 permutation.
...to get to termination.

$n! \geq (\frac{n}{e})^n \implies \log n! = \Omega(\log(n^n))$

Sorting lower bound: ...proof

Algorithm must be able to output any of $n!$ permuations.
Algorithm must output just 1 permutation at termination.
S is set of possible permutations at some point in Algorithm

Do some comparision: $a_i > a_j$?

If Yes, Alg "could" return subset of permutations: S_1.
If No, Alg "could" return subset of permutations: S_2.

$S_1 \cup S_2$ has to be S.

One of S_1 or S_2 contains at least half of S.

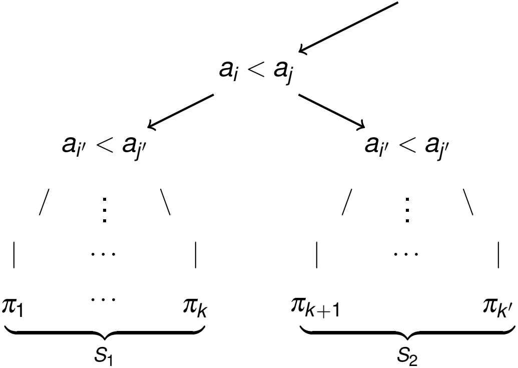
Consider larger of S_1 or S_2 and next comparision ...
... divides size of output set by at most 2.

Each comparision divides possible outputs by at most 2.

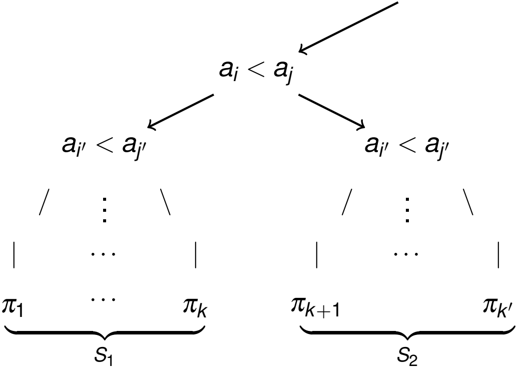
Need at least $\log_2(n!)$ comparisions to get to just 1 permutation.
...to get to termination.

$n! \geq (\frac{n}{e})^n \implies \log n! = \Omega(\log(n^n)) = \Omega(n \log n)$. $\square$

A figure proof.

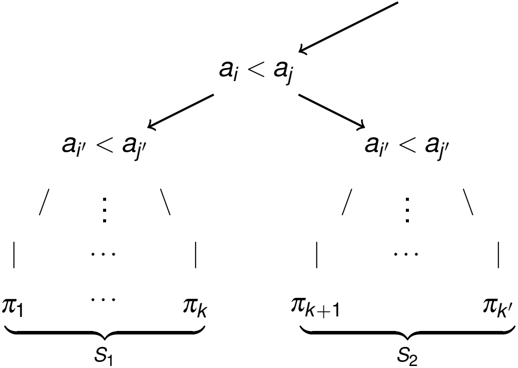


A figure proof.



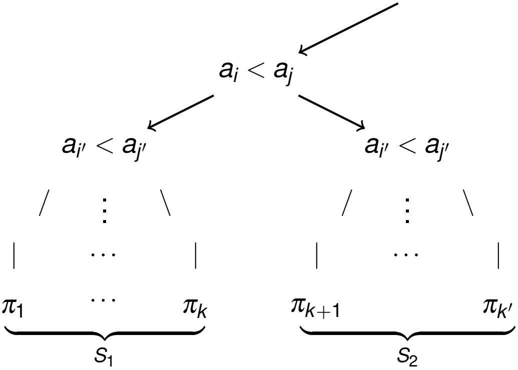
Either the set of permutations S_1 or S_2 is larger.

A figure proof.



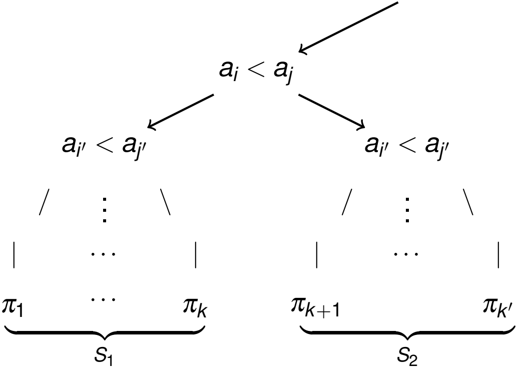
Either the set of permutations S_1 or S_2 is larger.
One must be at least half.

A figure proof.



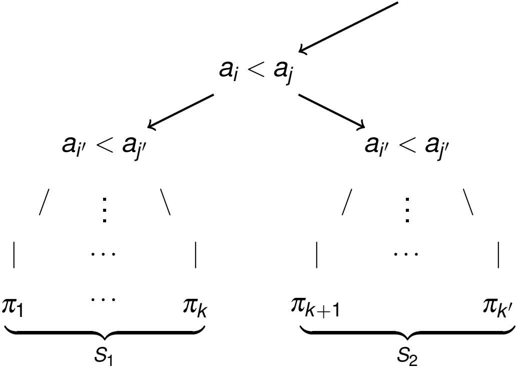
Either the set of permutations S_1 or S_2 is larger.
One must be at least half.
Depth must be $O(\log(\#\mathit{permutations}))$

A figure proof.



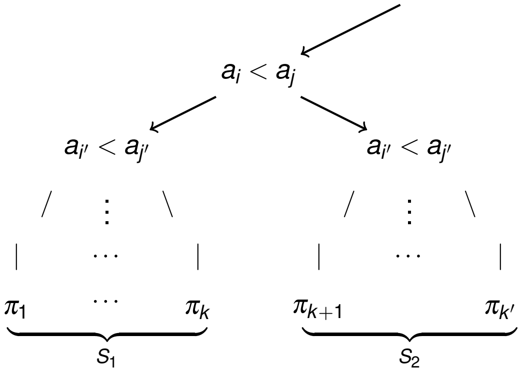
Either the set of permutations S_1 or S_2 is larger.
One must be at least half.
Depth must be $O(\log(\#\textit{permutations})) = O(\log n!)$

A figure proof.



Either the set of permutations S_1 or S_2 is larger.
One must be at least half.
Depth must be $O(\log(\#permutations)) = O(\log n!) = O(n \log n)$.

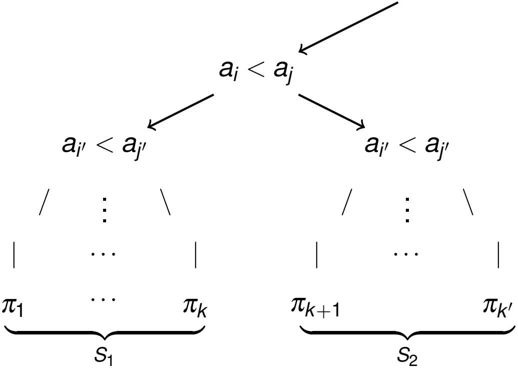
A figure proof.



Either the set of permutations S_1 or S_2 is larger.
One must be at least half.
Depth must be $O(\log(\#permutations)) = O(\log n!) = O(n \log n)$.

Can we do better than mergesort?

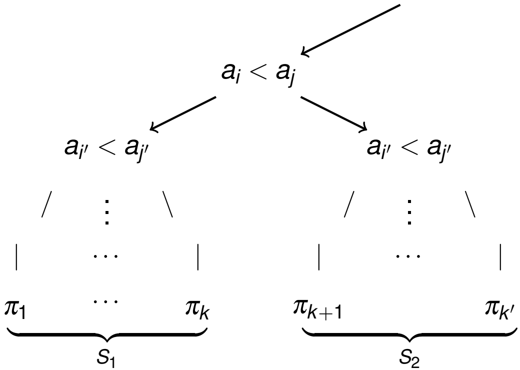
A figure proof.



Either the set of permutations S_1 or S_2 is larger.
One must be at least half.
Depth must be $O(\log(\#permutations)) = O(\log n!) = O(n \log n)$.

Can we do better than mergesort? Yes?

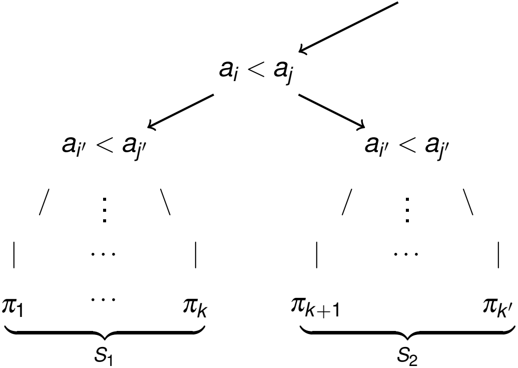
A figure proof.



Either the set of permutations S_1 or S_2 is larger.
One must be at least half.
Depth must be $O(\log(\#permutations)) = O(\log n!) = O(n \log n)$.

Can we do better than mergesort? Yes? No?

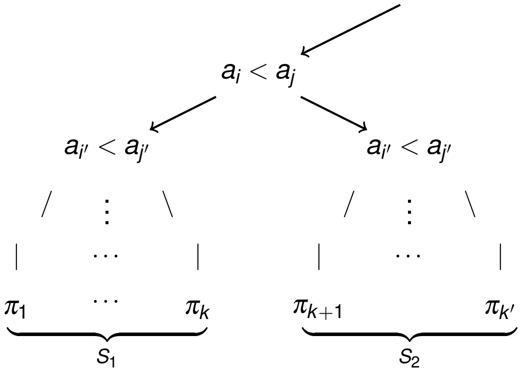
A figure proof.



Either the set of permutations S_1 or S_2 is larger.
One must be at least half.
Depth must be $O(\log(\#permutations)) = O(\log n!) = O(n \log n)$.

Can we do better than mergesort? Yes? No? No. For comparision sort.

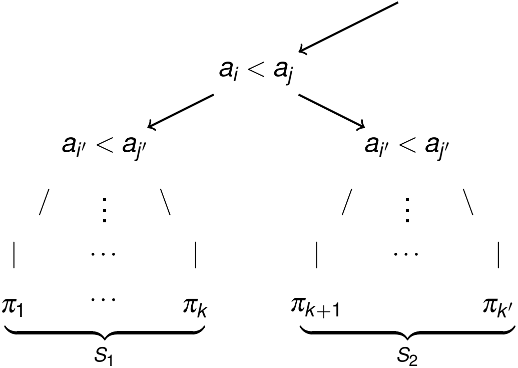
A figure proof.



Either the set of permutations S_1 or S_2 is larger.
One must be at least half.
Depth must be $O(\log(\#permutations)) = O(\log n!) = O(n \log n)$.

Can we do better than mergesort? Yes? No? No. For comparision sort.
(Recall from 61b: radix sort may be faster: $O(n)$.)

A figure proof.



Either the set of permutations S_1 or S_2 is larger.
One must be at least half.
Depth must be $O(\log(\#permutations)) = O(\log n!) = O(n \log n)$.

Can we do better than mergesort? Yes? No? No. For comparision sort.
(Recall from 61b: radix sort may be faster: $O(n)$.)

A bag of worms... “bit complexity” versus “word complexity”.

Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.

Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.

Median is value, v, where $\frac{n}{2}$ elts are less than v (if n is odd.)

Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.

Median is value, v, where $\frac{n}{2}$ elts are less than v (if n is odd.)

Versus Average?

Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.

Median is value, v, where $\frac{n}{2}$ elts are less than v (if n is odd.)

Versus Average?

Average household income (2004): \$70,700

Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.

Median is value, v, where $\frac{n}{2}$ elts are less than v (if n is odd.)

Versus Average?

Average household income (2004): \$70,700
Median household income (2004): \$43,200

Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.

Median is value, v, where $\frac{n}{2}$ elts are less than v (if n is odd.)

Versus Average?

Average household income (2004): \$70,700
Median household income (2004): \$43,200

Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.

Median is value, v, where $\frac{n}{2}$ elts are less than v (if n is odd.)

Versus Average?

Average household income (2004): $70,700
Median household income (2004): $43,200

Why so different?

Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.

Median is value, v, where $\frac{n}{2}$ elts are less than v (if n is odd.)

Versus Average?

Average household income (2004): \$70,700
Median household income (2004): \$43,200

Why so different? Bill Gates.

Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.

Median is value, v, where $\frac{n}{2}$ elts are less than v (if n is odd.)

Versus Average?

Average household income (2004): $70,700
Median household income (2004): $43,200

Why so different? Bill Gates. The 1%, perhaps.

Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.

Median is value, v, where $\frac{n}{2}$ elts are less than v (if n is odd.)

Versus Average?

Average household income (2004): \$70,700
Median household income (2004): \$43,200

Why so different? Bill Gates. The 1%, perhaps.

Why use average?

Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.

Median is value, v, where $\frac{n}{2}$ elts are less than v (if n is odd.)

Versus Average?

Average household income (2004): $70,700
Median household income (2004): $43,200

Why so different? Bill Gates. The 1%, perhaps.

Why use average?

Find average?

Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.

Median is value, v, where $\frac{n}{2}$ elts are less than v (if n is odd.)

Versus Average?

Average household income (2004): $70,700
Median household income (2004): $43,200

Why so different? Bill Gates. The 1%, perhaps.

Why use average?

Find average? Compute $\frac{\sum_i a_i}{n}$.

Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.

Median is value, v, where $\frac{n}{2}$ elts are less than v (if n is odd.)

Versus Average?

Average household income (2004): \$70,700
Median household income (2004): \$43,200

Why so different? Bill Gates. The 1%, perhaps.

Why use average?

Find average? Compute $\frac{\sum_i a_i}{n}$.

$O(n)$ time.

Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.

Median is value, v, where $\frac{n}{2}$ elts are less than v (if n is odd.)

Versus Average?

Average household income (2004): $70,700
Median household income (2004): $43,200

Why so different? Bill Gates. The 1%, perhaps.

Why use average?

Find average? Compute $\frac{\sum_i a_i}{n}$.

$O(n)$ time.

Compute median?

Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.

Median is value, v, where $\frac{n}{2}$ elts are less than v (if n is odd.)

Versus Average?

Average household income (2004): \$70,700
Median household income (2004): \$43,200

Why so different? Bill Gates. The 1%, perhaps.

Why use average?

Find average? Compute $\frac{\sum_i a_i}{n}$.

$O(n)$ time.

Compute median? Sort to get $s_1, \ldots s_n$.

Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.

Median is value, v, where $\frac{n}{2}$ elts are less than v (if n is odd.)

Versus Average?

Average household income (2004): \$70,700
Median household income (2004): \$43,200

Why so different? Bill Gates. The 1%, perhaps.

Why use average?

Find average? Compute $\frac{\sum_i a_i}{n}$.

$O(n)$ time.

Compute median? Sort to get $s_1, \ldots s_n$. Output element $s_{n/2+1}$.

Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.

Median is value, v, where $\frac{n}{2}$ elts are less than v (if n is odd.)

Versus Average?

Average household income (2004): $70,700
Median household income (2004): $43,200

Why so different? Bill Gates. The 1%, perhaps.

Why use average?

Find average? Compute $\frac{\sum_i a_i}{n}$.

$O(n)$ time.

Compute median? Sort to get $s_1, \ldots s_n$. Output element $s_{n/2+1}$.

$O(n \log n)$ time.

Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.

Median is value, v, where $\frac{n}{2}$ elts are less than v (if n is odd.)

Versus Average?

Average household income (2004): \$70,700
Median household income (2004): \$43,200

Why so different? Bill Gates. The 1%, perhaps.

Why use average?

Find average? Compute $\frac{\sum_i a_i}{n}$.

$O(n)$ time.

Compute median? Sort to get $s_1, \ldots s_n$. Output element $s_{n/2+1}$.

$O(n \log n)$ time.

Better algorithm?

Solve a harder Problem: Selection.

Solve a harder Problem: Selection.

For a set of n items S.

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Example.
$k = 7$ for items $\{11, 48, 5, 21, 2, 15, 17, 19, 15\}$

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Example.
$k = 7$ for items $\{11, 48, 5, 21, 2, 15, 17, 19, 15\}$

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Example.
$k = 7$ for items $\{11, 48, 5, 21, 2, 15, 17, 19, 15\}$

Output?

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Example.
$k = 7$ for items $\{11, 48, 5, 21, 2, 15, 17, 19, 15\}$

Output?

(A) 19
(B) 15
(C) 21

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Example.
$k = 7$ for items $\{11, 48, 5, 21, 2, 15, 17, 19, 15\}$

Output?

(A) 19
(B) 15
(C) 21

????

Solve a harder Problem: Selection.

Solve a harder Problem: Selection.

For a set of n items S.

Solve a harder Problem: Selection.

For a set of n items S.

Select kth smallest element.

Solve a harder Problem: Selection.

For a set of n items S.

Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Select(k, S)**:** $k = 7$ $S: 11, 48, 5, 21, 2, 15, 17, 19, 15$

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Select(k, S)**:** $k = 7$ $S : 11, 48, 5, 21, 2, 15, 17, 19, 15$

Base Case: $k = 1$ and $|S| = 1$, return elt.

Choose rand. elt b from A. $v = 15$

Solve a harder Problem: Selection.

For a set of n items S.

Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Select(k, S)**:** $k = 7$ $S: 11, 48, 5, 21, 2, 15, 17, 19, 15$

Base Case: $k = 1$ and $|S| = 1$, return elt.

Choose rand. elt b from A. $v = 15$

Form S_L containing all elts $< v$

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Select(k, S)**:** $k = 7$ $S : 11, 48, 5, 21, 2, 15, 17, 19, 15$

Base Case: $k = 1$ and $|S| = 1$, return elt.

Choose rand. elt b from A. $v = 15$

Form S_L containing all elts $< v$ $S_L : 11, 5, 2$

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Select(k, S)**:** $k = 7$ $S : 11, 48, 5, 21, 2, 15, 17, 19, 15$

Base Case: $k = 1$ and $|S| = 1$, return elt.

Choose rand. elt b from A. $v = 15$

Form S_L containing all elts $< v$ $S_L : 11, 5, 2$

Form S_v containing all elts $= v$

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Select(k, S)**:** $k = 7$ $S: 11, 48, 5, 21, 2, 15, 17, 19, 15$

Base Case: $k = 1$ and $|S| = 1$, return elt.

Choose rand. elt b from A. $v = 15$

Form S_L containing all elts $< v$ $S_L: 11, 5, 2$

Form S_v containing all elts $= v$ $S_v: 15, 15$

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Select(k, S)**:** $k = 7$ $S : 11, 48, 5, 21, 2, 15, 17, 19, 15$

Base Case: $k = 1$ and $|S| = 1$, return elt.

Choose rand. elt b from A. $v = 15$

Form S_L containing all elts $< v$ $S_L : 11, 5, 2$

Form S_v containing all elts $= v$ $S_v : 15, 15$

Form S_R containing all elts $> v$

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Select(k, S)**:** $k = 7$ $S : 11, 48, 5, 21, 2, 15, 17, 19, 15$

Base Case: $k = 1$ and $|S| = 1$, return elt.
Choose rand. elt b from A. $v = 15$
Form S_L containing all elts $< v$ $S_L : 11, 5, 2$
Form S_v containing all elts $= v$ $S_v : 15, 15$
Form S_R containing all elts $> v$ $S_R : 48, 21, 17, 19$

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Select(k, S)**:** $k = 7$ $S : 11, 48, 5, 21, 2, 15, 17, 19, 15$

Base Case: $k = 1$ and $|S| = 1$, return elt.

Choose rand. elt b from A. $v = 15$

Form S_L containing all elts $< v$ $S_L : 11, 5, 2$

Form S_v containing all elts $= v$ $S_v : 15, 15$

Form S_R containing all elts $> v$ $S_R : 48, 21, 17, 19$

If $k \leq |S_L|$, Select(k, S_L).

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Select(k, S)**:** $k = 7$ $S : 11, 48, 5, 21, 2, 15, 17, 19, 15$

Base Case: $k = 1$ and $|S| = 1$, return elt.

Choose rand. elt b from A. $v = 15$

Form S_L containing all elts $< v$ $S_L : 11, 5, 2$

Form S_v containing all elts $= v$ $S_v : 15, 15$

Form S_R containing all elts $> v$ $S_R : 48, 21, 17, 19$

If $k \leq |S_L|$, Select(k, S_L).
elseif $k \leq |S_L| + |S_v|$, return v.

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Select(k, S)**:** $k = 7$ $S : 11, 48, 5, 21, 2, 15, 17, 19, 15$

Base Case: $k = 1$ and $|S| = 1$, return elt.
Choose rand. elt b from A. $v = 15$
Form S_L containing all elts $< v$ $S_L : 11, 5, 2$
Form S_v containing all elts $= v$ $S_v : 15, 15$
Form S_R containing all elts $> v$ $S_R : 48, 21, 17, 19$

If $k \leq |S_L|$, Select(k, S_L).
elseif $k \leq |S_L| + |S_v|$, return v.
else Select$(k - |S_L| - |S_v|, S_R)$

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Select(k, S)**:** $k = 7$ $S : 11, 48, 5, 21, 2, 15, 17, 19, 15$

Base Case: $k = 1$ and $|S| = 1$, return elt.

Choose rand. elt b from A. $v = 15$

Form S_L containing all elts $< v$ $S_L : 11, 5, 2$

Form S_v containing all elts $= v$ $S_v : 15, 15$

Form S_R containing all elts $> v$ $S_R : 48, 21, 17, 19$

If $k \leq |S_L|$, Select(k, S_L).
elseif $k \leq |S_L| + |S_v|$, return v.
else Select$(k - |S_L| - |S_v|, S_R)$ Select$(2, [48, 21, 17, 19])$

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Select(k, S)**:** $k = 7$ $S: 11, 48, 5, 21, 2, 15, 17, 19, 15$

Base Case: $k = 1$ and $|S| = 1$, return elt.
Choose rand. elt b from A. $v = 15$
Form S_L containing all elts $< v$ $S_L: 11, 5, 2$
Form S_v containing all elts $= v$ $S_v: 15, 15$
Form S_R containing all elts $> v$ $S_R: 48, 21, 17, 19$

If $k \leq |S_L|$, Select(k, S_L).
elseif $k \leq |S_L| + |S_v|$, return v.
else Select$(k - |S_L| - |S_v|, S_R)$ Select$(2, [48, 21, 17, 19])$

Will eventually return 19, which is 7th element of list.

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Select(k, S)**:** $k = 7$ $S: 11, 48, 5, 21, 2, 15, 17, 19, 15$

Base Case: $k = 1$ and $|S| = 1$, return elt.
Choose rand. elt b from A. $v = 15$
Form S_L containing all elts $< v$ $S_L: 11, 5, 2$
Form S_v containing all elts $= v$ $S_v: 15, 15$
Form S_R containing all elts $> v$ $S_R: 48, 21, 17, 19$

If $k \leq |S_L|$, Select(k, S_L).
elseif $k \leq |S_L| + |S_v|$, return v.
else Select$(k - |S_L| - |S_v|, S_R)$ Select$(2, [48, 21, 17, 19])$

Will eventually return 19, which is 7th element of list.

Correctness:

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Select(k, S): $k = 7$ $S : 11, 48, 5, 21, 2, 15, 17, 19, 15$

Base Case: $k = 1$ and $|S| = 1$, return elt.

Choose rand. elt b from A. $v = 15$

Form S_L containing all elts $< v$ $S_L : 11, 5, 2$

Form S_v containing all elts $= v$ $S_v : 15, 15$

Form S_R containing all elts $> v$ $S_R : 48, 21, 17, 19$

If $k \leq |S_L|$, Select(k, S_L).
elseif $k \leq |S_L| + |S_v|$, return v.
else Select$(k - |S_L| - |S_v|, S_R)$ Select$(2, [48, 21, 17, 19])$

Will eventually return 19, which is 7th element of list.

Correctness: Induction.
Idea: Subroutine returns correct answer,

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Select(k, S)**:** $\quad k = 7$	$S: 11, 48, 5, 21, 2, 15, 17, 19, 15$
Base Case: $k = 1$ and $\lvert S \rvert = 1$, return elt.	
Choose rand. elt b from A.	$v = 15$
Form S_L containing all elts $< v$	$S_L: 11, 5, 2$
Form S_v containing all elts $= v$	$S_v: 15, 15$
Form S_R containing all elts $> v$	$S_R: 48, 21, 17, 19$
If $k \leq \lvert S_L \rvert$, Select(k, S_L).	
elseif $k \leq \lvert S_L \rvert + \lvert S_v \rvert$, return v.	
else Select$(k - \lvert S_L \rvert - \lvert S_v \rvert, S_R)$	Select$(2, [48, 21, 17, 19])$

Will eventually return 19, which is 7th element of list.

Correctness: Induction.
Idea: Subroutine returns correct answer, and

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Select(k, S)**:** $k = 7$ $S: 11, 48, 5, 21, 2, 15, 17, 19, 15$

Base Case: $k = 1$ and $|S| = 1$, return elt.
Choose rand. elt b from A. $v = 15$
Form S_L containing all elts $< v$ $S_L: 11, 5, 2$
Form S_v containing all elts $= v$ $S_v: 15, 15$
Form S_R containing all elts $> v$ $S_R: 48, 21, 17, 19$

If $k \leq |S_L|$, Select(k, S_L).
elseif $k \leq |S_L| + |S_v|$, return v.
else Select$(k - |S_L| - |S_v|, S_R)$ Select$(2, [48, 21, 17, 19])$

Will eventually return 19, which is 7th element of list.

Correctness: Induction.
Idea: Subroutine returns correct answer, and so

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Select(k, S): $k = 7$ $S: 11, 48, 5, 21, 2, 15, 17, 19, 15$

Base Case: $k = 1$ and $|S| = 1$, return elt.

Choose rand. elt b from A. $v = 15$

Form S_L containing all elts $< v$ $S_L: 11, 5, 2$

Form S_v containing all elts $= v$ $S_v: 15, 15$

Form S_R containing all elts $> v$ $S_R: 48, 21, 17, 19$

If $k \leq |S_L|$, Select(k, S_L).
elseif $k \leq |S_L| + |S_v|$, return v.
else Select$(k - |S_L| - |S_v|, S_R)$ Select$(2, [48, 21, 17, 19])$

Will eventually return 19, which is 7th element of list.

Correctness: Induction.
Idea: Subroutine returns correct answer, and so will

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Select(k, S)**:** $k = 7$ $S : 11, 48, 5, 21, 2, 15, 17, 19, 15$

Base Case: $k = 1$ and $|S| = 1$, return elt.
Choose rand. elt b from A. $v = 15$
Form S_L containing all elts $< v$ $S_L : 11, 5, 2$
Form S_v containing all elts $= v$ $S_v : 15, 15$
Form S_R containing all elts $> v$ $S_R : 48, 21, 17, 19$

If $k \leq |S_L|$, Select(k, S_L).
elseif $k \leq |S_L| + |S_v|$, return v.
else Select$(k - |S_L| - |S_v|, S_R)$ Select$(2, [48, 21, 17, 19])$

Will eventually return 19, which is 7th element of list.

Correctness: Induction.
Idea: Subroutine returns correct answer, and so will I

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Select(k, S)**:** $k = 7$ $S : 11, 48, 5, 21, 2, 15, 17, 19, 15$

Base Case: $k = 1$ and $|S| = 1$, return elt.

Choose rand. elt b from A. $v = 15$

Form S_L containing all elts $< v$ $S_L : 11, 5, 2$

Form S_v containing all elts $= v$ $S_v : 15, 15$

Form S_R containing all elts $> v$ $S_R : 48, 21, 17, 19$

If $k \leq |S_L|$, Select(k, S_L).
elseif $k \leq |S_L| + |S_v|$, return v.
else Select$(k - |S_L| - |S_v|, S_R)$ Select$(2, [48, 21, 17, 19])$

Will eventually return 19, which is 7th element of list.

Correctness: Induction.
Idea: Subroutine returns correct answer, and so will I !
Base case is good.

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Select(k, S)**:** $k = 7$ $S : 11, 48, 5, 21, 2, 15, 17, 19, 15$

Base Case: $k = 1$ and $|S| = 1$, return elt.

Choose rand. elt b from A. $v = 15$

Form S_L containing all elts $< v$ $S_L : 11, 5, 2$

Form S_v containing all elts $= v$ $S_v : 15, 15$

Form S_R containing all elts $> v$ $S_R : 48, 21, 17, 19$

If $k \leq |S_L|$, Select(k, S_L).
elseif $k \leq |S_L| + |S_v|$, return v.
else Select$(k - |S_L| - |S_v|, S_R)$ Select$(2, [48, 21, 17, 19])$

Will eventually return 19, which is 7th element of list.

Correctness: Induction.
Idea: Subroutine returns correct answer, and so will I !
Base case is good. Subroutine calls

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Select(k,S)**:** $k = 7$ $S: 11, 48, 5, 21, 2, 15, 17, 19, 15$

Base Case: $k = 1$ and $|S| = 1$, return elt.

Choose rand. elt b from A. $v = 15$

Form S_L containing all elts $< v$ $S_L : 11, 5, 2$

Form S_v containing all elts $= v$ $S_v : 15, 15$

Form S_R containing all elts $> v$ $S_R : 48, 21, 17, 19$

If $k \leq |S_L|$, Select(k, S_L).

elseif $k \leq |S_L| + |S_v|$, return v.

else Select$(k - |S_L| - |S_v|, S_R)$ Select$(2, [48, 21, 17, 19])$

Will eventually return 19, which is 7th element of list.

Correctness: Induction.

Idea: Subroutine returns correct answer, and so will I !

Base case is good. Subroutine calls ..by design.

Selection: runtime.

Worst case runtime?

Selection: runtime.

Worst case runtime?

(A) $\Theta(n \log n)$

(B) $\Theta(n)$

(C) $\Theta(n^2)$

Let $k = n$.

Selection: runtime.

Worst case runtime?

(A) $\Theta(n \log n)$
(B) $\Theta(n)$
(C) $\Theta(n^2)$

Let $k = n$.
Partition element is smallest every time.

Selection: runtime.

Worst case runtime?

(A) $\Theta(n \log n)$

(B) $\Theta(n)$

(C) $\Theta(n^2)$

Let $k = n$.
Partition element is smallest every time.

Size of list decrease by 1 in each recursive call.

Selection: runtime.

Worst case runtime?

(A) $\Theta(n \log n)$

(B) $\Theta(n)$

(C) $\Theta(n^2)$

Let $k = n$.
Partition element is smallest every time.

Size of list decrease by 1 in each recursive call.

Time for partition is $O(i)$ time when i elements.

Selection: runtime.

Worst case runtime?

(A) $\Theta(n \log n)$
(B) $\Theta(n)$
(C) $\Theta(n^2)$

Let $k = n$.
Partition element is smallest every time.

Size of list decrease by 1 in each recursive call.

Time for partition is $O(i)$ time when i elements.

$\Theta(n + (n-1) + \cdots + 2 + 1) = \Theta(n^2)$ time. or (C)

Selection: runtime.

Worst case runtime?

(A) $\Theta(n \log n)$

(B) $\Theta(n)$

(C) $\Theta(n^2)$

Let $k = n$.
Partition element is smallest every time.

Size of list decrease by 1 in each recursive call.

Time for partition is $O(i)$ time when i elements.

$\Theta(n + (n-1) + \cdots + 2 + 1) = \Theta(n^2)$ time. or (C)

Worse than sorting!

Selection: runtime.

Worst case runtime?

(A) $\Theta(n \log n)$

(B) $\Theta(n)$

(C) $\Theta(n^2)$

Let $k = n$.
Partition element is smallest every time.

Size of list decrease by 1 in each recursive call.

Time for partition is $O(i)$ time when i elements.

$\Theta(n + (n-1) + \cdots + 2 + 1) = \Theta(n^2)$ time. or (C)

Worse than sorting!

On average?

Average time to get a heads?

Flip a coin, what is average number of tosses to get a heads?

Average time to get a heads?

Flip a coin, what is average number of tosses to get a heads?

(A) two

(B) three

Average time to get a heads?

Flip a coin, what is average number of tosses to get a heads?

(A) two
(B) three
(C) Could go forever!

Average time to get a heads?

Flip a coin, what is average number of tosses to get a heads?

(A) two
(B) three
(C) Could go forever!

(A) ..

Average time to get a heads?

Flip a coin, what is average number of tosses to get a heads?

(A) two
(B) three
(C) Could go forever!

(A) ..and (C)

Average time to get a heads?

Flip a coin, what is average number of tosses to get a heads?

(A) two

(B) three

(C) Could go forever!

(A) ..and (C) (but not relevant.)

Expected (average) Time?

Expected (average) Time?

Lemma: Expected number of coin tosses to get a heads is 2.

Expected (average) Time?

Lemma: Expected number of coin tosses to get a heads is 2.

Proof: $E[X] = 1 + \frac{1}{2}E[X]$

Expected (average) Time?

Lemma: Expected number of coin tosses to get a heads is 2.

Proof: $E[X] = 1 + \frac{1}{2}E[X]$
$\implies \frac{1}{2}E[X] = 1$

Expected (average) Time?

Lemma: Expected number of coin tosses to get a heads is 2.

Proof: $E[X] = 1 + \frac{1}{2}E[X]$
$\implies \frac{1}{2}E[X] = 1 \implies E[X] = 2.$ $\square$

Expected (average) Time?

Lemma: Expected number of coin tosses to get a heads is 2.

Proof: $E[X] = 1 + \frac{1}{2}E[X]$
$\implies \frac{1}{2}E[X] = 1 \implies E[X] = 2.$ $\square$

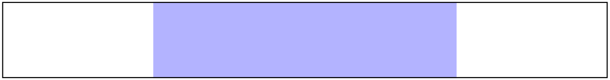
Probability that random elt in the middle half is $\geq \frac{1}{2}$.

Expected (average) Time?

Lemma: Expected number of coin tosses to get a heads is 2.

Proof: $E[X] = 1 + \frac{1}{2}E[X]$
$\implies \frac{1}{2}E[X] = 1 \implies E[X] = 2.$ $\square$

Probability that random elt in the middle half is $\geq \frac{1}{2}$.

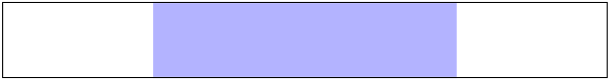


Expected (average) Time?

Lemma: Expected number of coin tosses to get a heads is 2.

Proof: $E[X] = 1 + \frac{1}{2}E[X]$
$\implies \frac{1}{2}E[X] = 1 \implies E[X] = 2.$ $\square$

Probability that random elt in the middle half is $\geq \frac{1}{2}$.



Expected time to get a middle element is $E[X] \times O(n) = O(n)$.

Expected (average) Time?

Lemma: Expected number of coin tosses to get a heads is 2.

Proof: $E[X] = 1 + \frac{1}{2}E[X]$
$\implies \frac{1}{2}E[X] = 1 \implies E[X] = 2.$ $\square$

Probability that random elt in the middle half is $\geq \frac{1}{2}$.

Expected time to get a middle element is $E[X] \times O(n) = O(n)$.

If pick in middle half subproblem size is $\leq \frac{3}{4}n$.

Expected (average) Time?

Lemma: Expected number of coin tosses to get a heads is 2.

Proof: $E[X] = 1 + \frac{1}{2}E[X]$
$\implies \frac{1}{2}E[X] = 1 \implies E[X] = 2.$ $\square$

Probability that random elt in the middle half is $\geq \frac{1}{2}$.

Expected time to get a middle element is $E[X] \times O(n) = O(n)$.

If pick in middle half subproblem size is $\leq \frac{3}{4}n$.

Expected time recurrence:

$$T(n) \leq$$

Expected (average) Time?

Lemma: Expected number of coin tosses to get a heads is 2.

Proof: $E[X] = 1 + \frac{1}{2}E[X]$
$\implies \frac{1}{2}E[X] = 1 \implies E[X] = 2.$ $\square$

Probability that random elt in the middle half is $\geq \frac{1}{2}$.

Expected time to get a middle element is $E[X] \times O(n) = O(n)$.

If pick in middle half subproblem size is $\leq \frac{3}{4}n$.

Expected time recurrence:

$$T(n) \leq T(\frac{3}{4}n) +$$

Expected (average) Time?

Lemma: Expected number of coin tosses to get a heads is 2.

Proof: $E[X] = 1 + \frac{1}{2}E[X]$
$\implies \frac{1}{2}E[X] = 1 \implies E[X] = 2.$ $\square$

Probability that random elt in the middle half is $\geq \frac{1}{2}$.

Expected time to get a middle element is $E[X] \times O(n) = O(n)$.

If pick in middle half subproblem size is $\leq \frac{3}{4}n$.

Expected time recurrence:

$$T(n) \leq T(\frac{3}{4}n) + O(n).$$

Expected (average) Time?

Lemma: Expected number of coin tosses to get a heads is 2.

Proof: $E[X] = 1 + \frac{1}{2}E[X]$
$\implies \frac{1}{2}E[X] = 1 \implies E[X] = 2.$ $\square$

Probability that random elt in the middle half is $\geq \frac{1}{2}$.

Expected time to get a middle element is $E[X] \times O(n) = O(n)$.

If pick in middle half subproblem size is $\leq \frac{3}{4}n$.

Expected time recurrence:

$$T(n) \leq T(\frac{3}{4}n) + O(n).$$

Masters or just thinking

Expected (average) Time?

Lemma: Expected number of coin tosses to get a heads is 2.

Proof: $E[X] = 1 + \frac{1}{2}E[X]$
$\implies \frac{1}{2}E[X] = 1 \implies E[X] = 2.$ $\square$

Probability that random elt in the middle half is $\geq \frac{1}{2}$.

Expected time to get a middle element is $E[X] \times O(n) = O(n)$.

If pick in middle half subproblem size is $\leq \frac{3}{4}n$.

Expected time recurrence:

$$T(n) \leq T(\frac{3}{4}n) + O(n).$$

Masters or just thinking $(n + (3/4)n + (3/4)^2 n + \cdots = O(n))$

Expected (average) Time?

Lemma: Expected number of coin tosses to get a heads is 2.

Proof: $E[X] = 1 + \frac{1}{2}E[X]$
$\implies \frac{1}{2}E[X] = 1 \implies E[X] = 2.$ $\square$

Probability that random elt in the middle half is $\geq \frac{1}{2}$.

Expected time to get a middle element is $E[X] \times O(n) = O(n)$.

If pick in middle half subproblem size is $\leq \frac{3}{4}n$.

Expected time recurrence:

$$T(n) \leq T(\frac{3}{4}n) + O(n).$$

Masters or just thinking $(n + (3/4)n + (3/4)^2 n + \cdots = O(n))$
$\implies T(n) = O(n)$.

See you ..

...on Wednesday..