Run Time Bounds

Upper bounds on the *Worst Case* running times of sorting algorithms:

- Bubblesort: $O(n^2)$
- Modified Bubblesort: $O(n^2)$
- Insertion sort: $O(n^2)$
- Mergesort $O(n \log(n))$
- Heapsort $O(n \log(n))$
- Quicksort $O(n^2)$

Run Time Bounds cont.

Lower bounds on the *Worst Case* running times of sorting algorithms:

- Bubblesort: $\Omega(n^2)$
- Modified Bubblesort: $\Omega(n^2)$
- Insertion sort: $\Omega(n^2)$
- Mergesort: $\Omega(n \log(n))$
- Heapsort: $\Omega(n \log(n))$
- Quicksort: $\Omega(n^2)$

Insertion Sort

```
Insertion sort(S)
    For i=2 to n
        For j=i-1 downto 1
            If s_j>s_{j+1} then swap(s_j, s_{j+1})
```


Bubblesort

```
Bubblesort(S)
    Assume S={s_1,s_2, ... ,s_n}
    For i=n down to 2
        For j=1 to i-1
            If s_j>s_{j+1} then swap(s_j, s_{j+1})
    Return
```


Modified Bubblesort

```
Modified-Bubblesort(S)
    Swapped=T
    For i=n down to 2
        If Swapped=F then return
        Swapped=F
        For j=1 to i-1
            If s_j>s_{j+1} then swap(s_j, s_{j+1}) and set
              Swapped=T
    Return
```


Heapsort

```
Heapsort(S)
    H = Buildheap(S)
    For i=n  downto 1 {
        s_i = root(H)
        H = remove-root(H)
        }
    Return(S)
```

Quicksort

Quicksort (S)

Choose an s from S as pivot

S_1= {t in S such that t≤s}

S_2 = {t in S such that t>s}

Return Quicksort(S_1), s, Quicksort(S_2)

Mergesort

Mergesort (S={s_1, s_2, ... , s_n)

If n=1 return(S)

Else

S_1 = Mergesort(s_1, ... , $s_{n/2}$)

S_2 = Mergesort($s_{n/2+1}$, ... , s_n)

Return Merge(S_1 , S_2)

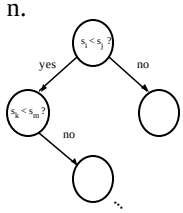
Lower Bound for Sorting

Theorem: Any comparison-based sorting algorithms has a worst-case running time that is Ω(n log(n)).

A comparison-based algorithm is one that doesn't need to read the input provided it is given the size of the input and a comparison oracle.

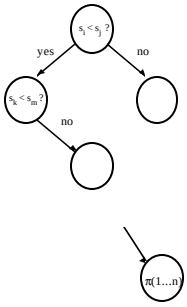
Proof of Theorem

A comparison based algorithm can be represented by a decision tree for each input size n.



Each root to leaf path represents the sequence of queries for a particular input.

Proof of Theorem cont.



Each leaf corresponds to the permutation the algorithm has decided sorts the input

Proof cont.

- There must be at least n! leaves.
- A binary tree with n! leaves has some path of length at least log(n!)
- By Stirling's approximation:

  log(n!)=Ω(n log(n))

Linear Time Sorting

Can we do it?

Yes – but not by "comparison-based" algorithms

Some examples:

- Counting Sort
- Radix Sort