

Lecture 25: Time Efficiency of Algorithms (con't)

The efficiency of an algorithm *may* depend on the data.

ex. Do a linear search for the number 18 in array A.

Best case:

Worst case:

Average case:

The efficiency of some algorithms do NOT depend on the data.

To judge efficiency we can time the algorithms or count operations. (Operation =

Examples of counting operations:

Number of Operations

Big-O (really Θ)

1) for (int j = 0; j < N; j++)
 X = X * X + 2 * Y;

2) for (int j = 0; j < N; j++)
 for (int i = 0; i < N; i++)
 T = T + 2 * R - Y;

3) for (int j = 0; j < N; j++)
 for (int i = 0; i <= j; i++)
 if (A[i] > 100)
 A[i] = 10;

Here the number of operations varies. Why?

It is customary to count the number of comparisons.

j =	0	1	2			...	
Times thru i loop							

Each time through the i loop, _____ comparisons are performed

Total Number of Comparisons =

So Big-O:

```

4)   for (int j = 1; j < N; j++)
      for (int i = 0; i < N - j; i++)
        if (A[i] > A[i + 1])
          {
            Temp = A[i];
            A[i] = A[i + 1];
            A[i + 1] = Temp;
          }

```

j =	1	2	3			...	
Times thru i loop							

Total Number of Comparisons

Big-O

```

5)   for (int j = 1; j <= N2; j++)
      for (int i = 1; i <= j; i++)
        if (A[j] == A[i])
          T = T + 3;

```

j =	1	2	3			...	
Times thru i loop							

Number of Comparisons

Big-O

When the Efficiency Depends on the Data—Some Classic Algorithms

Linear Search

```
i ← 0  
found ← false  
while not found  
  if A[i] == Target  
  
  else
```

Best Case:

Worst Case:

Average Case:

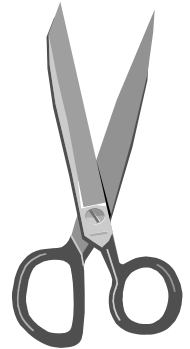
Binary Search

```
First ← 0  
Last ← N - 1  
found ← false  
  
while not found and First != Last  
  Mid ← (First + Last) / 2  
  if A[Mid] == Target  
  
  else if A[Mid] > Target  
  
  else  
  
if First == Last  
  if A[First] == Target  
  
  else
```

Best Case:

Worst Case:

Average Case:



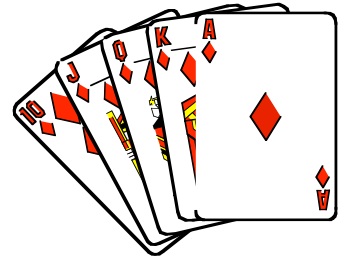
Insertion Sort

```
for i = 1 to N-1
  Temp ← A[i]
  j ← i - 1
  while j >= 0 and A[j] > Temp
    A[j + 1] = A[j] //slide
    j ← j - 1
  A[j+1] ← Temp // stuff
```

Best Case

Worst Case

Average Case



Merge Sort

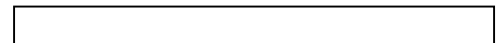


Time(N) =

Does the time taken by this algorithm depend on the data?

QuickSort

```
Pick a Pivot point
move all elements < Pivot to the left and elements >= Pivot to the right
    (call these Sets I and II)
move the Pivot to a position separating Sets I and II
QuickSort Set I
QuickSort Set II
```



If Sets I and II are equal in size Time(N) =

Does the time taken by this algorithm depend on the data?