INTRODUCTION & RECAP

Design and Analysis of Algorithms

Design and Analysis of Algorithms

- Design

Design and Analysis of Algorithms

- Design
- Analysis
- Proving the correctness of the algorithm
- measuring the efficiency of the algorithm
    - How much of the resources [space, time, ..] needed?

Design and Analysis of Algorithms

- Design
- Analysis
- Proving the correctness of the algorithm
- measuring the efficiency of the algorithm
  - How much of the resources [space, time, ..] needed?

Design and Analysis of Algorithms

- Design
- Analysis
- Proving the correctness of the algorithm
- measuring the efficiency of the algorithm
  - How much of the resources [space, time, ..] needed?
  - worst- case running time as a function of input size

Linear Search

Input : A[0, …, n-1] Array of n element, key

Output : location of key in A, if present. Otherwise -1

```
for i=0 to n-1
  if A[i] = key
    return i
return -1
```

Binary Search

Input : A[0, …, n-1] Ordered Array of n elements, key

Output : location of key in A, if present. Otherwise -1

```
first=0; last = n-1
mid = (first+last)/2
while first ≤ last
   if A[mid]=key
      return mid
   else if A[mid] < key
      first =mid +1
   else last = mid-1
return -1
```

Bubble Sort

Input : Array of n elements, A

Output : A in sorted order

```
for i=1 to n-1
  for j=i+1 to n
    if A[i] > A[j]
      swap A[i],A[j]
```

Asymptotic Notation

Represents how fast a function grows

Asymptotic Notation

Big - Oh notation

Asymptotic Notation

Big - Oh notation

- represents the set of functions that are upper bounded by $g(n)$

Asymptotic Notation

Big - Oh notation

O(g(n)) = $\{ f(n) \mid \exists c > 0, n_0 \geq 0 \ s.t \ 0 \leq f(n) \leq cg(n), \forall n \geq n_0 \}$

- represents the set of functions that are upper bounded by g(n)

Asymptotic Notation

Big - Omega notation

- represents the set of functions that are lower bounded by g(n)

Asymptotic Notation

Big - Omega notation

$$\Omega(g(n)) = \{ f(n) \mid \exists c > 0, n_0 \geq 0 \ s.t \ 0 \leq cg(n) \leq f(n), \forall n \geq n_0 \}$$

- represents the set of functions that are lower bounded by g(n)

Asymptotic Notation

Theta notation

$f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

Asymptotic Notation

Theta notation

$f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

$\Theta(g(n))$ =

$\{$

$f(n) \mid \exists c_1, c_2 > 0, n_0 \geq 0 \ s.t \ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0$

$\}$

Asymptotic Notation

o - notation

to denote an upper bound that is not asymptotically tight

$$o(g(n)) = \{ f(n) \mid \forall c > 0, \exists n_0 \geq 0 \ s.t \ 0 \leq f(n) < cg(n), \forall n \geq n_0 \}$$

Asymptotic Notation

ω- notation

to denote a lower bound that is not asymptotically tight

$\omega(g(n))$ =

$$\{ f(n) \mid \forall c > 0, \exists n_0 \geq 0 \ s.t \ 0 \leq cg(n) < f(n), \forall n \geq n_0 \}$$

Asymptotic Notation

Properties

- Transitivity
- Reflexivity
- Symmetry
- Transpose Symmetry

Proving the correctness

Linear Search

Input : A[0, …, n-1] Array of n element, key

Output : location of key in A, if present. Otherwise -1

```
for i=0 to n-1
  if A[i] = key
    return i
return -1
```

Linear Search

Input : A[0, . . ., n-1] Array of n element, key

Output : location of key in A, if present. Otherwise -1

```
for i=0 to n-1
  if A[i] = key
    return i
return -1
```

Searches the entire solution space.

Binary Search

Input : A[0, …, n-1] Ordered Array of n elements, key

Output : location of key in A, if present. Otherwise -1

```
first=0; last = n-1

mid = (first+last)/2

while first ≤ last

   if A[mid]=key

      return mid

   else if A[mid] < key

      first =mid +1

   else last = mid-1

return -1
```

Binary Search

Input : A[0, . . ., n-1] Ordered Array of n elements, key

Output : location of key in A, if present. Otherwise -1

```
first=0; last = n-1

mid = (first+last)/2

while first ≤ last

   if A[mid]=key

      return mid

   else if A[mid] < key

      first =mid +1

   else last = mid-1

return -1
```

Induction on number of elements in the array

Bubble Sort

Input : Array of n elements, A

Output : A in sorted order

```
for i=1 to n-1
  for j=i+1 to n
    if A[i] > A[j]
      swap A[i],A[j]
```

Bubble Sort

Input : Array of n elements, A

Output : A in sorted order

```
for i=1 to n-1
  for j=i+1 to n
    if A[i] > A[j]
      swap A[i],A[j]
```

For all i, after i iterations, A[1 ….i] contains the i smallest elements in the right order.

By induction on i.

Insertion Sort

Input : Array of n elements, arr

Output : arr in sorted order

```
for i= 1 to n

        key = arr[i];
        j = i - 1;

        while (j ≥ 0 and arr[j] > key)

            arr[j + 1] = arr[j];
            j = j - 1;

        arr[j + 1] = key;
```