MCAC 301: Design and Analysis of Algorithms

Neelima Gupta

ngupta@cs.du.ac.in

August 24, 2020

Sorting

Definition

Sorting: Given a partial order set $(X, \leq)$, problem is to arrange the elements in X so that $x_1 \leq x_2 \leq .... \leq x_n$.

Insertion Sort

input : Array: $A[1], A[2] ..., A[n]$
output: Sorted array; $A[1] \leq A[2] \leq ..... \leq A[n]$

for *i: 2 to n* **do**
  Insert $A[i]$ in the partially sorted array $A[1 \ldots i-1]$
end

$$A[1] \leq a[2] \ldots \leq A[i-1], \; A[i] \ldots A[n]$$

Insertion Sort

input : Array: $A[1], A[2]..., A[n]$
output: Sorted array; $A[1] \leq A[2] \leq ..... \leq A[n]$

for *i: 2 to n* **do**
    $j = Find - location(A, i)$ /* function returns the location where $A[i]$ must be inserted so that the array $A[1 \ldots i]$ is sorted */

j=3

10, 20, 30, 40, 25, 8, 35, 32

i=5

Insertion Sort

input : Array: $A[1], A[2]..., A[n]$
output: Sorted array; $A[1] \leq A[2] \leq ..... \leq A[n]$

for *i: 2 to n* **do**
  $j = Find - location(A, i)$ /* function returns the location where $A[i]$ must be inserted so that the array $A[1 \ldots i]$ is sorted */
  $Insert(A, i, j)$ /* function inserts $A[i]$ in the j^{th} location. This involves shifting elements to the right
end

$\underbrace{10,\ 20,\ 25,\ 30,\ 40,}\ 8,\ 35,\ 32$

Insertion Sort

input : Array: $A[1], A[2] ..., A[n]$
output: Sorted array; $A[1] \leq A[2] \leq ..... \leq A[n]$

for *i: 2 to n* **do**
  $j = Find - location(A, i)$ /* function returns the location where $A[i]$ must be inserted so that the array $A[1 \ldots i]$ is sorted */
  $Insert(A, i, j)$ /* function inserts $A[i]$ in the j^{th} location. This involves shifting elements to the right
end

Let $T(n)$ be the number of primitive steps performed by Insertion Sort on n numbers. Then,

Insertion Sort

---

input : Array: $A[1], A[2]..., A[n]$
output: Sorted array; $A[1] \leq A[2] \leq ..... \leq A[n]$

---

for *i: 2 to n* **do**

$j = Find - location(A, i)$ /* function returns the location where $A[i]$ must be inserted so that the array $A[1 \ldots i]$ is sorted */

TfI(i)

$Insert(A, i, j)$ /* function inserts $A[i]$ in the j^{th} location. This involves shifting elements to the right

TIns(i,j)

end

Let $T(n)$ be the number of primitive steps performed by Insertion Sort on n numbers. Then, $T(n) = \sum_i TfI(i) + \sum_i TIns(i, j)$ where $TfI()$ and $TIns(,)$ are the number of number of primitive steps performed by $Find - location()$ and $Insert()$ respectively.

Find-location(A, i)

input : Sorted array $A[1] \leq A[2] \leq ..... \leq A[i-1]$ and $A[i]$
output: index j: $A[j] \leq A[i] < A[j+1]$

$key = A[i]$
$j = i - 1$
while $j > 0$ && $A[j] > key$ **do**
  $j = j - 1$
end
return $j + 1$

10, 20, 30, 40, 25 8, 35, 32

$i=5$

$A[2] < 25$
return 3

Find-location(A, i)

input : Sorted array $A[1] \leq A[2] \leq ..... \leq A[i-1]$ and $A[i]$
output: index j: $A[j] \leq A[i] < A[j+1]$

$key = A[i]$
$j = i - 1$
while $j > 0$ && $A[j] > key$ **do** ← t_i
    $j = j - 1$
end
return $j + 1$

What is the value of t_i in the worst case?

Find-location(A, i)

input : Sorted array $A[1] \leq A[2] \leq ..... \leq A[i-1]$ and $A[i]$
output: index j: $A[j] \leq A[i] < A[j+1]$

$key = A[i]$
$j = i - 1$
while $j > 0$ && $A[j] > key$ **do** $\leftarrow t_i$
  $j = j - 1$
end
return $j + 1$

What is the value of t_i in the worst case? i, right?

Find-location(A, i)

input : Sorted array $A[1] \leq A[2] \leq ..... \leq A[i-1]$ and $A[i]$
output: index j: $A[j] \leq A[i] < A[j+1]$

$key = A[i]$
$j = i - 1$
while $j > 0$ && $A[j] > key$ **do** $\leftarrow t_i$
  $j = j - 1$
end
return $j + 1$

What is the value of t_i in the worst case? i, right? $(i-1)+1$.

Find-location(A, i)

input : Sorted array $A[1] \leq A[2] \leq ..... \leq A[i-1]$ and $A[i]$
output: index j: $A[j] \leq A[i] < A[j+1]$

$key = A[i]$
$j = i - 1$
while $j > 0$ && $A[j] > key$ **do** $\leftarrow t_i$
  $j = j - 1$
end
return $j + 1$

What is the value of t_i in the worst case? i, right? $(i-1)+1$.
What is the value of t_i in the best case?

Find-location(A, i)

input : Sorted array $A[1] \leq A[2] \leq ..... \leq A[i-1]$ and $A[i]$
output: index j: $A[j] \leq A[i] < A[j+1]$

$key = A[i]$
$j = i - 1$
while $j > 0$ && $A[j] > key$ **do** ← t_i
  $j = j - 1$
end
return $j + 1$

What is the value of t_i in the worst case? i, right? $(i-1)+1$.
What is the value of t_i in the best case? 1, right?

Find-location(A, i)

input : Sorted array $A[1] \leq A[2] \leq ..... \leq A[i-1]$ and $A[i]$
output: index j: $A[j] \leq A[i] < A[j+1]$

$key = A[i]$
$j = i - 1$
while $j > 0 \ \&\& \ A[j] > key$ **do** $\leftarrow t_i$
  $j = j - 1$
end
return $j + 1$

What is the value of t_i in the worst case? i, right? $(i-1)+1$.
What is the value of t_i in the best case? 1, right?
Number of times the while-statement is executed = the number of times the key comparison results in a success +1.

Find-location(A, i)

input : Sorted array $A[1] \leq A[2] \leq ..... \leq A[i-1]$ and $A[i]$
output: index j: $A[j] \leq A[i] < A[j+1]$

$key = A[i]$
$j = i - 1$
while $j > 0 \,\&\&\, A[j] > key$ **do** $\longleftarrow t_i$
  $j = j - 1$
end
return $j + 1$

What is the value of t_i in the worst case? i, right? $(i-1)+1$.
What is the value of t_i in the best case? 1, right?
Number of times the while-statement is executed $=$ the number of times the key comparison results in a success $+1$. Thus,
$1 \leq Tfl(i) \leq i$.

Insert(i, j)

input : Sorted array $A[1] \leq A[2] \leq ..... \leq A[i-1]$ and $A[i]$
output: Sorted array; $A[1] \leq A[2] \leq ..... \leq A[i]$ inserting $A[i]$ in the j^{th} location.

$k = i - 1$
while $k > j - 1$ **do**
  $A[k+1] = A[k]$
  $k = k - 1$
end
$A[k+1] = key$

$j=3$ $i=5$

10, 20, 30, 40, 25, 8, 35, 32

10, 20, 25, 30, 40, 8, 35, 32

Insert(i, j)

---

input : Sorted array $A[1] \leq A[2] \leq ..... \leq A[i-1]$ and $A[i]$
output: Sorted array; $A[1] \leq A[2] \leq ..... \leq A[i]$ inserting $A[i]$ in the j^{th} location.

$k = i - 1$
while $k > j - 1$ **do** $\longleftarrow t_{ij}$
  $A[k+1] = A[k]$
  $k = k - 1$
end
$A[k+1] = key$

What is the value of t_{ij}?

Insert(i, j)

input : Sorted array $A[1] \leq A[2] \leq ..... \leq A[i-1]$ and $A[i]$
output: Sorted array; $A[1] \leq A[2] \leq ..... \leq A[i]$ inserting $A[i]$ in the j^{th} location.

$k = i - 1$
while $k > j - 1$ **do** ⟵ t_{ij}
  $A[k+1] = A[k]$
  $k = k - 1$
end
$A[k+1] = key$

What is the value of t_{ij}? $(i-1) - (j-1) + 1)$ right?

Insert(i, j)

input : Sorted array $A[1] \leq A[2] \leq ..... \leq A[i-1]$ and $A[i]$
output: Sorted array; $A[1] \leq A[2] \leq ..... \leq A[i]$ inserting $A[i]$ in the j^{th} location.

$k = i - 1$
while $k > j - 1$ **do** $\longleftarrow t_{ij}$
  $A[k+1] = A[k]$
  $k = k - 1$
end
$A[k+1] = key$

What is the value of t_{ij}? $(i-1)-(j-1)+1)$ right? i.e. $i - j + 1$.

Insert(i, j)

input : Sorted array $A[1] \leq A[2] \leq ..... \leq A[i-1]$ and $A[i]$
output: Sorted array; $A[1] \leq A[2] \leq ..... \leq A[i]$ inserting $A[i]$ in the j^{th} location.

$k = i - 1$
while $k > j - 1$ **do** ← t_{ij}
    $A[k+1] = A[k]$
    $k = k - 1$
end
$A[k+1] = key$

What is the value of t_{ij}? $(i-1)-(j-1)+1)$ right? i.e. $i - j + 1$.
which in the worst case is?

Insert(i, j)

input : Sorted array $A[1] \leq A[2] \leq ..... \leq A[i-1]$ and $A[i]$
output: Sorted array; $A[1] \leq A[2] \leq ..... \leq A[i]$ inserting $A[i]$ in the j^{th} location.

$k = i - 1$
while $k > j - 1$ **do** ← t_{ij}
    $A[k+1] = A[k]$
    $k = k - 1$
end
$A[k+1] = key$

What is the value of t_{ij}? $(i-1) - (j-1) + 1)$ right? i.e. $i - j + 1$.
which in the worst case is? i, right?

Insert(i, j)

input : Sorted array $A[1] \le A[2] \le \ldots.. \le A[i-1]$ and $A[i]$
output: Sorted array; $A[1] \le A[2] \le \ldots.. \le A[i]$ inserting $A[i]$ in the j^{th} location.

$k = i - 1$
while $k > j - 1$ **do** t_{ij}
    $A[k+1] = A[k]$
    $k = k - 1$
end
$A[k+1] = key$

What is the value of t_{ij}? $(i-1)-(j-1)+1)$ right? i.e. $i - j + 1$.
which in the worst case is? i, right? When?

Insert(i, j)

input : Sorted array $A[1] \leq A[2] \leq ..... \leq A[i-1]$ and $A[i]$
output: Sorted array; $A[1] \leq A[2] \leq ..... \leq A[i]$ inserting $A[i]$ in the j^{th} location.

$k = i - 1$
while $k > j - 1$ **do** $\longleftarrow t_{ij}$
    $A[k+1] = A[k]$
    $k = k - 1$
end
$A[k+1] = key$

What is the value of t_{ij}? $(i-1) - (j-1) + 1)$ right? i.e.
$i - j + 1$.
which in the worst case is? i, right? When? when $j = 1$, right?

Insert(i, j)

input : Sorted array $A[1] \leq A[2] \leq ..... \leq A[i-1]$ and $A[i]$
output: Sorted array; $A[1] \leq A[2] \leq ..... \leq A[i]$ inserting $A[i]$ in the j^{th} location.

$k = i - 1$
while $k > j - 1$ **do** $\leftarrow t_{ij}$
    $A[k+1] = A[k]$
    $k = k - 1$
end
$A[k+1] = key$

What is the value of t_{ij}? $(i-1) - (j-1) + 1)$ right? i.e. $i - j + 1$.
which in the worst case is? i, right? When? when $j = 1$, right?
and in the best case?

Insert(i, j)

input : Sorted array $A[1] \leq A[2] \leq ..... \leq A[i-1]$ and $A[i]$
output: Sorted array; $A[1] \leq A[2] \leq ..... \leq A[i]$ inserting $A[i]$ in the j^{th} location.

$k = i - 1$
while $k > j - 1$ **do** ← t_{ij}
  $A[k+1] = A[k]$
  $k = k - 1$
end
$A[k+1] = key$

What is the value of t_{ij}? $(i-1) - (j-1) + 1)$ right? i.e. $i - j + 1$.
which in the worst case is? i, right? When? when $j = 1$, right?
and in the best case? 1, right?

Insert(i, j)

input : Sorted array $A[1] \leq A[2] \leq ..... \leq A[i-1]$ and $A[i]$
output: Sorted array; $A[1] \leq A[2] \leq ..... \leq A[i]$ inserting $A[i]$ in the j^{th} location.

$k = i - 1$
while $k > j - 1$ **do** $\longleftarrow t_{ij}$
  $A[k+1] = A[k]$
  $k = k - 1$
end
$A[k+1] = key$

What is the value of t_{ij}? $(i-1) - (j-1) + 1)$ right? i.e. $i - j + 1$.
which in the worst case is? i, right? When? when $j = 1$, right?
and in the best case? 1, right?

Insert(i, j)

input : Sorted array $A[1] \leq A[2] \leq ..... \leq A[i-1]$ and $A[i]$
output: Sorted array; $A[1] \leq A[2] \leq ..... \leq A[i]$ inserting $A[i]$ in the j^{th} location.

$k = i - 1$
while $k > j - 1$ **do** ← t_{ij}
  $A[k+1] = A[k]$
  $k = k - 1$
end
$A[k+1] = key$

What is the value of t_{ij}? $(i-1)-(j-1)+1)$ right? i.e. $i - j + 1$.
which in the worst case is? i, right? When? when $j = 1$, right?
and in the best case? 1, right? When?

Insert(i, j)

input : Sorted array $A[1] \leq A[2] \leq ..... \leq A[i-1]$ and $A[i]$
output: Sorted array; $A[1] \leq A[2] \leq ..... \leq A[i]$ inserting $A[i]$ in the j^{th} location.

$k = i - 1$
while $k > j - 1$ **do** t_{ij}
  $A[k+1] = A[k]$
  $k = k - 1$
end
$A[k+1] = key$

What is the value of t_{ij}? $(i-1)-(j-1)+1)$ right? i.e. $i - j + 1$.
which in the worst case is? i, right? When? when $j = 1$, right?
and in the best case? 1, right? When? when $j = i$, right?

Insert(i, j)

input : Sorted array $A[1] \leq A[2] \leq ..... \leq A[i-1]$ and $A[i]$
output: Sorted array; $A[1] \leq A[2] \leq ..... \leq A[i]$ inserting $A[i]$ in the j^{th} location.

$k = i - 1$
while $k > j - 1$ **do** ⟵ t_{ij}
  $A[k+1] = A[k]$
  $k = k - 1$
end
$A[k+1] = key$

What is the value of t_{ij}? $(i-1) - (j-1) + 1)$ right? i.e. $i - j + 1$.
which in the worst case is? i, right? When? when $j = 1$, right?
and in the best case? 1, right? When? when $j = i$, right?
Number of times the while-statement is executed $=$ the number of assignments $+1$.

Insert(i, j)

input : Sorted array $A[1] \le A[2] \le ..... \le A[i-1]$ and $A[i]$
output: Sorted array; $A[1] \le A[2] \le ..... \le A[i]$ inserting $A[i]$ in the j^{th} location.

$k = i - 1$
while $k > j - 1$ **do** ⟵ t_{ij}
    $A[k+1] = A[k]$
    $k = k - 1$
end
$A[k+1] = key$

What is the value of t_{ij}? $(i-1) - (j-1) + 1)$ right? i.e. $i - j + 1$.
which in the worst case is? i, right? When? when $j = 1$, right?
and in the best case? 1, right? When? when $j = i$, right?
Number of times the while-statement is executed = the number of assignments $+1$. Thus, $1 \le TIns(i, j) \le i$.

What can be dropped/ignored while counting the primitive steps

- Loop Control Variables

What can be dropped/ignored while counting the primitive steps

- Loop Control Variables
- Statements that are executed constant number of times

What can be dropped/ignored while counting the primitive steps

Linear Search Revisited:

---

for $i \leftarrow 1$ *to* n **do**
    if $A[i] = key$ **then**
        return i
    end
end
return 0

What can be dropped/ignored while counting the primitive steps

Linear Search Revisited:

for $i \leftarrow 1$ *to* n **do**
    if $A[i] = key$ **then**
        return i
    end
end
return 0

We just ask "How many times the "If" statement is executed?"

What can be dropped/ignored while counting the primitive steps

Linear Search Revisited:

---

for $i \leftarrow 1$ *to* n **do**
    if $A[i] = key$ **then**
        return i
    end
end
return 0

We just ask "How many times the "If" statement is executed?" Because the number of times "for-statement" is executed is same as this (plus minus some constant) and hence can be absorbed in a multiplicative factor of 2.

What can be dropped/ignored while counting the primitive steps

Linear Search Revisited:

---

for $i \leftarrow 1$ *to* n **do**
    if $A[i] = key$ **then**
        return i
    end
end
return 0

We just ask "How many times the "If" statement is executed?" Because the number of times "for-statement" is executed is same as this (plus minus some constant) and hence can be absorbed in a multiplicative factor of 2. Return statement is executed only once and contributes only an additive constant to the number of primitive steps.

What can be dropped/ignored while counting the primitive steps

Linear Search Revisited:

---

for $i \leftarrow 1$ *to* n **do**
    if $A[i] = key$ **then**
        return i
    end
end
return 0

We just ask "How many times the "If" statement is executed?" Because the number of times "for-statement" is executed is same as this (plus minus some constant) and hence can be absorbed in a multiplicative factor of 2. Return statement is executed only once and contributes only an additive constant to the number of primitive steps. $2T2 + 2$.

What can be dropped/ignored while counting the primitive steps

Linear Search Revisited:

---

for $i \leftarrow 1$ *to* n **do**
  if $A[i] = key$ **then**
    return i
  end
end
return 0

We just ask "How many times the "If" statement is executed?" Because the number of times "for-statement" is executed is same as this (plus minus some constant) and hence can be absorbed in a multiplicative factor of 2. Return statement is executed only once and contributes only an additive constant to the number of primitive steps. $2T2 + 2.$

Ignoring the constants, we only need to count the number of key - comparisons.

What can be dropped/ignored while counting the primitive steps

Linear Search Revisited:

---

for $i \leftarrow 1$ *to* n **do**
  if $A[i] = key$ **then**
    return i
  end
end
return 0

We just ask "How many times the "If" statement is executed?" Because the number of times "for-statement" is executed is same as this (plus minus some constant) and hence can be absorbed in a multiplicative factor of 2. Return statement is executed only once and contributes only an additive constant to the number of primitive steps. $2T2 + 2$.

Ignoring the constants, we only need to count the number of key - comparisons. We further drop "key" and simply say the number of comparisons.

Find-location(A, i) revisited

$key = A[i]$
$j = i - 1$
while $j > 0$ && $A[j] > key$ **do**
    $j = j - 1$
end
return $j + 1$

Again, the number of times "while-statement" is executed is same as the number of "key"-comparisons (plus minus some constant) and hence is absorbed in a multiplicative factor of 2 of the number of times the key is compared". Return statement : Additive constant

Find-location(A, i) revisited

$key = A[i]$

$j = i - 1$

while $j > 0$ && $A[j] > key$ **do**

$\quad j = j - 1$

end

return $j + 1$

Again, the number of times "while-statement" is executed is same as the number of "key"-comparisons (plus minus some constant) and hence is absorbed in a multiplicative factor of 2 of the number of times the key is compared". Return statement : Additive constant

Thus, as before, Ignoring the constants, we only need to count the number of key - comparisons.

Insert(i, j) revisited

$k = i - 1$
while $k > j - 1$ **do**
  $A[k+1] = A[k]$
  $k = k - 1$
end
$A[k+1] = key$

We just ask "How many times the shift/move/assignment is done?" The number of times "while-statement" is executed is same as this and hence can be absorbed in a multiplicative factor of 2. Plus an additive constant due to the first and the last statement

Insert(i, j) revisited

$k = i - 1$

while $k > j - 1$ **do**

  $A[k+1] = A[k]$

  $k = k - 1$

end

$A[k+1] = key$

We just ask "How many times the shift/move/assignment is done?" The number of times "while-statement" is executed is same as this and hence can be absorbed in a multiplicative factor of 2. Plus an additive constant due to the first and the last statement Thus, as before, ignoring the constants, we only need to count the number of shifts/moves/assignments.

Insertion Sort : Analysis

for *i: 2 to n* **do**

$j = Find - location(A, i)$ /* function returns the location where $A[i]$ must be inserted so that the array $A[1 \dots i]$ is sorted */

$Insert(A, i, j)$ /* function inserts $A[i]$ in the j^{th} location. This involves shifting elements to the right

end

$T(n) = \sum_i TfI(i) + \sum_i TIns(i, j)$ within constant factors. Where $TfI()$ is the number of comparisons done by $Find - location()$ and $TIns(,)$ is the number of shifts/moves/assignments performed by $Insert()$.

Insertion Sort : Analysis

---

for *i: 2 to n* **do**

  $j = Find - location(A, i)$ /* function returns the location where $A[i]$ must be inserted so that the array $A[1 \ldots i]$ is sorted */

  $Insert(A, i, j)$ /* function inserts $A[i]$ in the j^{th} location. This involves shifting elements to the right

end

$T(n) = \sum_i TfI(i) + \sum_i TIns(i, j)$ within constant factors. Where $TfI()$ is the number of comparisons done by $Find - location()$ and $TIns(,)$ is the number of shifts/moves/assignments performed by $Insert()$.

Thus the time complexity of Insertion sort is determined by the number of key-comparisons and the number of shift operations it performs.

Insertion Sort : Analysis

Thus, in the worst case

$$T(n) = \sum_{i=1 \ to \ n-1} i + \sum_{i=1 \ to \ n-1} i = \theta(\frac{n(n-1)}{2})$$

Insertion Sort : Analysis

Thus, in the worst case

$$T(n) = \sum_{i=1 \ to \ n-1} i + \sum_{i=1 \ to \ n-1} i = \theta(\frac{n(n-1)}{2})$$

And, in the best case

$$T(n) = \sum_{i=1 \ to \ n-1} 1 + \sum_{i=1 \ to \ n-1} 1 = \theta(n)$$

Practically, a better implementation of Insertion Sort : Shift as you Compare

input : Array: $A[1], A[2] ..., A[n]$
output: Sorted array; $A[1] \leq A[2] \leq ..... \leq A[n]$

for *i: 2 to n* **do**
    $key = A[i]$
    $j = i - 1$
    while $j > 0 \ \&\& \ A[j] > key$ **do**
        $A[j+1] = A[j]$
        $j = j - 1$
    end
    $A[j+1] = key$
end

No need to run the loop twice.

Programming Assignment

Implement Insertion Sort

- Count the number of key comparisons and assignments for various inputs and plot the graph for both of them.
- For every input size n , run it with 10 different data points generated randomly.
- Compute the minimum, maximum and average of the number of key comparisons (and assignments) for each input size.
- Plot the graph for each case - best, worst and average number of comparisons (and assignments) .
- n varies from 10 to 100 in steps of 5.