Principles of Computer Science II

Sorting Algorithms

Ioannis Chatzigiannakis

Sapienza University of Rome

Lecture 8

Selection Sort Algorithm

This algorithm first finds the smallest element in the array and exchanges it with the element in the first position, then find the second smallest element and exchange it with the element in the second position, and continues in this way until the entire array is sorted.

Selection Sort: Example

Original Array	After 1st pass	After 2nd pass	After 3rd pass	After 4th pass	After 5th pass
3	1	1	1	1	1
6	6	3	3	3	3
1	3	6	4	4	4
8	8	8	8	5	5
4	4	4	6	6	6
5	5	5	5	8	8

Selection Sort Code

```
a = [5, 1, 6, 2, 4, 3]
for i in range(0, len(a)):
    min = i
    for j in range(i + 1, len(a) - 1):
        if a[j] > a[min]:
            min = j

    temp = a[j]
    a[j] = a[min]
    a[min] = temp
```

How good is Insertion Sort?

- How many comparisons are required until the list is sorted?

How good is Insertion Sort?

- How many comparisons are required until the list is sorted?
  - 1^{st} loop: n - 1
  - 2^{nd} loop: n - 2
  - ...

How good is Insertion Sort?

- How many comparisons are required until the list is sorted?
  - 1^{st} loop: n - 1
  - 2^{nd} loop: n - 2
  - ...
  - (n-1)+(n-2)+(n-3)+ ...+3+2+1 comparisons are required

How good is Insertion Sort?

- How many comparisons are required until the list is sorted?
  - 1^{st} loop: n - 1
  - 2^{nd} loop: n - 2
  - ...
  - (n-1)+(n-2)+(n-3)+ ...+3+2+1 comparisons are required
  - $\sum \frac{n(n-1)}{2}$ comparisons are required

How good is Insertion Sort?

- How many comparisons are required until the list is sorted?
  - 1^{st} loop: n - 1
  - 2^{nd} loop: n - 2
  - ...
  - (n-1)+(n-2)+(n-3)+ ...+3+2+1 comparisons are required
  - $\sum \frac{n(n-1)}{2}$ comparisons are required
- How much memory is needed ?

How good is Insertion Sort?

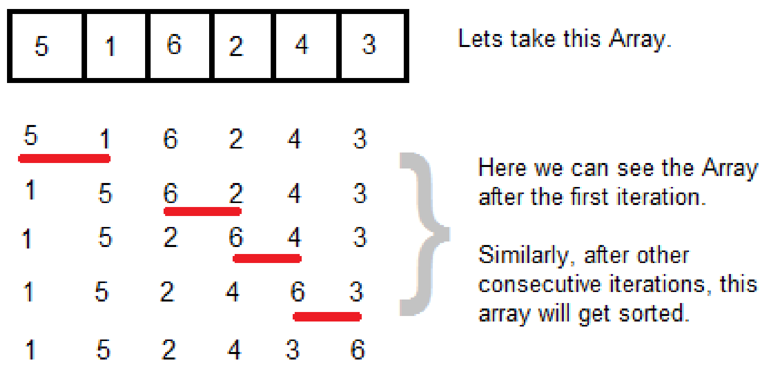
- How many comparisons are required until the list is sorted?
  - 1^{st} loop: n - 1
  - 2^{nd} loop: n - 2
  - ...
  - (n-1)+(n-2)+(n-3)+ ...+3+2+1 comparisons are required
  - $\sum \frac{n(n-1)}{2}$ comparisons are required
- How much memory is needed ?
  - 1 additional slot.

Bubble Sort Algorithm

Bubble Sort is an algorithm which is used to sort N elements that are given in a memory. Bubble Sort compares all the element one by one and sort them based on their values.

- It is called Bubble sort, because with each iteration the largest element in the list bubbles up towards the last place, just like a water bubble rises up to the water surface.
- Sorting takes place by stepping through all the data items one-by-one in pairs and comparing adjacent data items and swapping each pair that is out of order.

Bubble Sorting: Example



Bubble Sort Code

```
a = [5, 1, 6, 2, 4, 3]
for i in range(0, len(a)):
    for j in range(0, len(a) - i - 1):
        if a[j] > a[j+1]:
            temp = a[j]
            a[j] = a[j+1]
            a[j+1] = temp
```

- The above algorithm is not efficient because as per the above logic, the for-loop will keep executing for six iterations even if the list gets sorted after the second iteration.

Bubble Sort Code: Version 2

- We can insert a flag and can keep checking whether swapping of elements is taking place or not in the following iteration.
- If no swapping is taking place, it means the list is sorted and we can jump out of the for loop, instead executing all the iterations.

```
a = [5, 1, 6, 2, 4, 3]
for i in range(0, len(a)):
    for j in range(0, len(a) - i - 1):
        if a[j] > a[j+1]:
            temp = a[j]
            a[j] = a[j+1]
            a[j+1] = temp
```

How good is Bubble Sort?

- How many comparisons are required until the list is sorted?

How good is Bubble Sort?

- How many comparisons are required until the list is sorted?
  - 1^{st} loop: n - 1
  - 2^{nd} loop: n - 2
  - ...
  - (n-1)+(n-2)+(n-3)+ ...+3+2+1 comparisons are required
  - $\Sigma \frac{n(n-1)}{2}$ comparisons are required

How good is Bubble Sort?

- How many comparisons are required until the list is sorted?
  - 1^{st} loop: n - 1
  - 2^{nd} loop: n - 2
  - ...
  - (n-1)+(n-2)+(n-3)+ ...+3+2+1 comparisons are required
  - $\Sigma \frac{n(n-1)}{2}$ comparisons are required
- Is there a "simple" case ?

How good is Bubble Sort?

- How many comparisons are required until the list is sorted?
  - 1^{st} loop: n - 1
  - 2^{nd} loop: n - 2
  - ...
  - (n-1)+(n-2)+(n-3)+ ...+3+2+1 comparisons are required
  - $\Sigma \frac{n(n-1)}{2}$ comparisons are required
- Is there a "simple" case ?
  - How many loops are required?

How good is Bubble Sort?

- How many comparisons are required until the list is sorted?
  - 1^{st} loop: n - 1
  - 2^{nd} loop: n - 2
  - ...
  - (n-1)+(n-2)+(n-3)+ ...+3+2+1 comparisons are required
  - $\Sigma \frac{n(n-1)}{2}$ comparisons are required
- Is there a "simple" case ?
  - How many loops are required?
  - The list is already sorted

How good is Bubble Sort?

- How many comparisons are required until the list is sorted?
  - 1^{st} loop: n - 1
  - 2^{nd} loop: n - 2
  - ...
  - (n-1)+(n-2)+(n-3)+ ...+3+2+1 comparisons are required
  - $\Sigma \frac{n(n-1)}{2}$ comparisons are required
- Is there a "simple" case ?
  - How many loops are required?
  - The list is already sorted
  - N comparisons are required

How good is Bubble Sort?

- How many comparisons are required until the list is sorted?
  - 1^{st} loop: n - 1
  - 2^{nd} loop: n - 2
  - ...
  - (n-1)+(n-2)+(n-3)+ ...+3+2+1 comparisons are required
  - $\sum \frac{n(n-1)}{2}$ comparisons are required
- Is there a "simple" case ?
  - How many loops are required?
  - The list is already sorted
  - N comparisons are required
- How much memory is needed ?

How good is Bubble Sort?

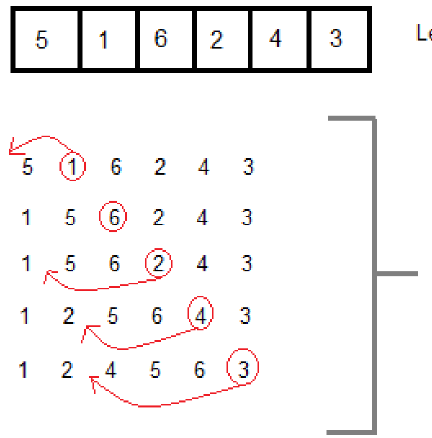
- How many comparisons are required until the list is sorted?
  - 1^{st} loop: n - 1
  - 2^{nd} loop: n - 2
  - ...
  - (n-1)+(n-2)+(n-3)+ ...+3+2+1 comparisons are required
  - $\sum \frac{n(n-1)}{2}$ comparisons are required
- Is there a "simple" case ?
  - How many loops are required?
  - The list is already sorted
  - N comparisons are required
- How much memory is needed ?
  - 1 additional slot.

Insert Sort Algorithm

A simple Sorting algorithm which sorts the list by shifting elements one by one.

- It has one of the simplest implementation
- It is efficient for smaller data sets, but very inefficient for larger lists.
- Insertion Sort is adaptive, that means it reduces its total number of steps if given a partially sorted list, hence it increases its efficiency.
- It is better than Selection Sort and Bubble Sort algorithms.
- Like Bubble Sorting, insertion sort also requires a single additional memory space.

Insertion Sort: Example



Lets take this Array.

As we can see here, in insertion sort, we pick up a key, and compares it with elemnts ahead of it, and puts the key in the right place

5 has nothing before it.

1 is compared to 5 and is inserted before 5.

6 is greater than 5 and 1.

2 is smaller than 6 and 5, but greater than 1, so its is inserted after 1.

And this goes on...

(Always we start with the second element as key.)

Insertion Sort Code

```
a = [5, 1, 6, 2, 4, 3]
for i in range(1, len(a)):
    key = a[i]
    j = i - 1
    while j >= 0 and key < a[j]:
        a[j+1] = a[j]
        j -= 1
    a[j+1] = key
```

- key: we put each element of the list, in each pass, starting from the second element: a[1].
- using the while loop, we iterate, until j becomes equal to zero or we find an element which is greater than key, and then we insert the key at that position.

How good is Insertion Sort?

- How many comparisons are required until the list is sorted?

How good is Insertion Sort?

- How many comparisons are required until the list is sorted?
  - 1^{st} loop: n - 1
  - 2^{nd} loop: n - 2
  - ...
  - (n-1)+(n-2)+(n-3)+ ...+3+2+1 comparisons are required
  - $\sum \frac{n(n-1)}{2}$ comparisons are required

How good is Insertion Sort?

- How many comparisons are required until the list is sorted?
  - 1^{st} loop: n - 1
  - 2^{nd} loop: n - 2
  - ...
  - (n-1)+(n-2)+(n-3)+ ...+3+2+1 comparisons are required
  - $\sum \frac{n(n-1)}{2}$ comparisons are required
- Is there a "simple" case ?

How good is Insertion Sort?

- How many comparisons are required until the list is sorted?
  - 1^{st} loop: n - 1
  - 2^{nd} loop: n - 2
  - ...
  - (n-1)+(n-2)+(n-3)+ ...+3+2+1 comparisons are required
  - $\Sigma\frac{n(n-1)}{2}$ comparisons are required
- Is there a "simple" case ?
  - The list is already sorted
  - N comparisons are required

How good is Insertion Sort?

- How many comparisons are required until the list is sorted?
  - 1^{st} loop: n - 1
  - 2^{nd} loop: n - 2
  - ...
  - (n-1)+(n-2)+(n-3)+ ...+3+2+1 comparisons are required
  - $\Sigma\frac{n(n-1)}{2}$ comparisons are required
- Is there a "simple" case ?
  - The list is already sorted
  - N comparisons are required

- How much memory is needed ?

How good is Insertion Sort?

- How many comparisons are required until the list is sorted?
  - 1^{st} loop: n - 1
  - 2^{nd} loop: n - 2
  - ...
  - (n-1)+(n-2)+(n-3)+ ...+3+2+1 comparisons are required
  - $\Sigma\frac{n(n-1)}{2}$ comparisons are required
- Is there a "simple" case ?
  - The list is already sorted
  - N comparisons are required
- How much memory is needed ?

  - 1 additional slot.

Quick Sort Algorithm

Quick sort is very fast and requires very less additional space. It is based on the rule of Divide and Conquer. This algorithm divides the list into three main parts :

- Elements less than the Pivot element
- Pivot element(Central element)
- Elements greater than the pivot element

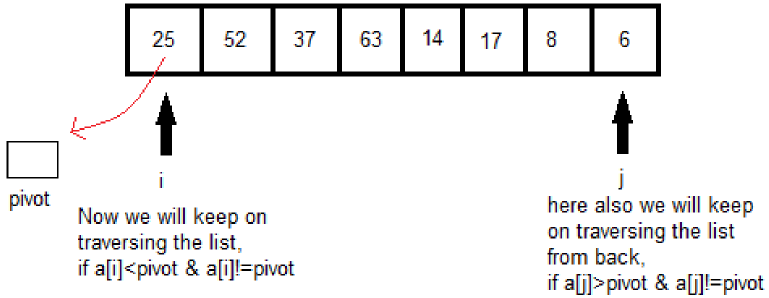
- Sorts any list very quickly
- Performance depends on the selection of the Pivot element

Quick Sort: Example

List: 25 52 37 63 14 17 8 6

- We pick 25 as the pivot.
- All the elements smaller to it on its left,
- All the elements larger than to its right.
- After the first pass the list looks like:
  6 8 17 14 25 63 37 52
- Now we sort two separate lists:
  6 8 17 14 and 63 37 52
- We apply the same logic, and we keep doing this until the complete list is sorted.

Quick Sort: Example



if both sides we find the element not satisfying their respective conditions, we swap them. And keep repeating this.

DIVIDE AND CONQUER - QUICK SORT

Quick Sort Code

```
a = [25, 52, 37, 63, 14, 17, 8, 6]

def partition(list, p, r):
    pivot = list[p]
    i = p
    j = r
    while(1):
        while(list[i] < pivot and list[i] != pivot):
            i += 1

        while(list[j] > pivot and list[j] != pivot):
            j -= 1

        if(i < j):
            temp = list[i]
            list[i] = list[j]
            list[j] = temp
        else:
            return j
```

Quick Sort Code

```
def quicksort(list, p, r):
    if (p < r):
        q = partition(list, p, r)
        quicksort(list, p, q);
        quicksort(list, q + 1, r);

print("Before: ", a)
quicksort(a, 0, len(a) - 1)
print("After: ", a)
```

How good is Quick Sort?

- How many comparisons are required until the list is sorted?

How good is Quick Sort?

- How many comparisons are required until the list is sorted?
- What if we choose the smallest or the largest item as pivot?

How good is Quick Sort?

- How many comparisons are required until the list is sorted?
- What if we choose the smallest or the largest item as pivot?
  - 1^{st} loop: n - 1
  - 2^{nd} loop: n - 2
  - ...
  - (n-1)+(n-2)+(n-3)+ ...+3+2+1 comparisons are required
  - $\sum \frac{n(n-1)}{2}$ comparisons are required

How good is Quick Sort?

- How many comparisons are required until the list is sorted?
- What if we choose the smallest or the largest item as pivot?
  - 1^{st} loop: n - 1
  - 2^{nd} loop: n - 2
  - ...
  - (n-1)+(n-2)+(n-3)+ ...+3+2+1 comparisons are required
  - $\sum \frac{n(n-1)}{2}$ comparisons are required
- What if we choose the median item as pivot?

How good is Quick Sort?

- How many comparisons are required until the list is sorted?
- What if we choose the smallest or the largest item as pivot?
  - 1^{st} loop: n - 1
  - 2^{nd} loop: n - 2
  - ...
  - (n-1)+(n-2)+(n-3)+ ...+3+2+1 comparisons are required
  - $\sum \frac{n(n-1)}{2}$ comparisons are required
- What if we choose the median item as pivot?
  - 1^{st} loop: two lists $\frac{n}{2}$ each
  - 2^{nd} loop: four lists $\frac{n}{4}$ each
  - ...
  - $\log n$ steps
  - For each partition we do n comparisons
  - In total $n \log n$ comparisons

How good is Quick Sort?

- How many comparisons are required until the list is sorted?
- What if we choose the smallest or the largest item as pivot?
  - 1^{st} loop: n - 1
  - 2^{nd} loop: n - 2
  - ...
  - (n-1)+(n-2)+(n-3)+ ...+3+2+1 comparisons are required
  - $\sum \frac{n(n-1)}{2}$ comparisons are required
- What if we choose the median item as pivot?
  - 1^{st} loop: two lists $\frac{n}{2}$ each
  - 2^{nd} loop: four lists $\frac{n}{4}$ each
  - ...
  - $\log n$ steps
  - For each partition we do n comparisons
  - In total $n \log n$ comparisons
- How much memory is needed ?

How good is Quick Sort?

- How many comparisons are required until the list is sorted?
- What if we choose the smallest or the largest item as pivot?
  - 1^{st} loop: n - 1
  - 2^{nd} loop: n - 2
  - ...
  - (n-1)+(n-2)+(n-3)+ ...+3+2+1 comparisons are required
  - $\sum \frac{n(n-1)}{2}$ comparisons are required
- What if we choose the median item as pivot?
  - 1^{st} loop: two lists $\frac{n}{2}$ each
  - 2^{nd} loop: four lists $\frac{n}{4}$ each
  - ...
  - $\log n$ steps
  - For each partition we do n comparisons
  - In total $n \log n$ comparisons
- How much memory is needed ?
  - 1 additional slot.