

Sorting algorithms

1&1



Presentation by Antonio-Gabriel Sturzu

- Algorithms based on key comparison
 - Selection sort
 - Insertion sort
 - Bubble sort
 - Merge Sort
 - Quick Sort

- At each step find the value of the minimum element
- Swap it with the first element in the current sublist
- Repeat the above steps for the rest of the list
- Example:
 - 5 4 3 2 1
 - 1 4 3 2 5
 - 1 2 3 4 5



Pseudocode:

- For $i=1, i \leq n, i++$
 - $imin=i$
 - For $j=i+1, j \leq n, j++$
 - if($x[j] < x[imin]$)
 - » $imin=j$
 - if($imin \neq i$)
 - » $swap(x[i], x[imin])$

- Time complexity is $\theta(n^2)$ on all cases because it always does $n*(n-1)/2$ comparisons



Main idea:

- Maintain an invariant which says that the first i elements of the array are sorted
- In order to maintain the invariant we must correctly insert the $i+1$ 'th element in the list formed by the first i elements such that all the $i+1$ elements will be correctly sorted

Example:

- 6 5 7 3 4 1 2
- 5 6 7 3 4 1 2
- 5 6 7 3 4 1 2
- 3 5 6 7 4 1 2
- 3 4 5 6 7 1 2
- 1 3 4 5 6 7 2
- 1 2 3 4 5 6 7



Pseudocode:

- For $i=2, i \leq n, i++$
 - $j=i, \text{save}=a[i]$
 - while($j>1 \ \&\& \ a[j-1]>\text{save}$)
 - $a[j]=a[j-1];$
 - $j=j-1$
 - $a[j]=\text{save}$

- The time complexity is $\theta(n+d)$ where d is the number of inversions of the array
- From this we can see that insertion sort will give very good results if the array is almost sorted
- The best case performance will be $\theta(n)$
- The problem is that the worst case and average case are $\theta(n^2)$
- The worst case occurs when the array is sorted in decreasing order

- In practice for sufficiently small arrays it is faster than merge sort and quick sort
- In many implementations insertion sort is used as a subroutine for merge sort or quick sort for sorting small arrays
- We shall see such an optimization in this presentation



Main idea:

- At each step of the algorithm go through the array and swap adjacent elements if they are not in the correct order
- Repeat this procedure until there will be no more swaps
- At that point we are sure that the array is already sorted



Example:

– 6 5 1 2 4 3

– First pass:

- 5 6 1 2 4 3

- 5 1 6 2 4 3

- 5 1 2 6 4 3

- 5 1 2 4 6 3

- 5 1 2 4 3 6



Example:

– 5 1 2 4 3 6

– Second pass:

- 1 5 2 4 3 6

- 1 2 5 4 3 6

- 1 2 4 5 3 6

- 1 2 4 3 5 6

Example:

- 1 2 4 3 5 6
- Third pass:
 - 1 2 3 4 5 6

Pseudocode:

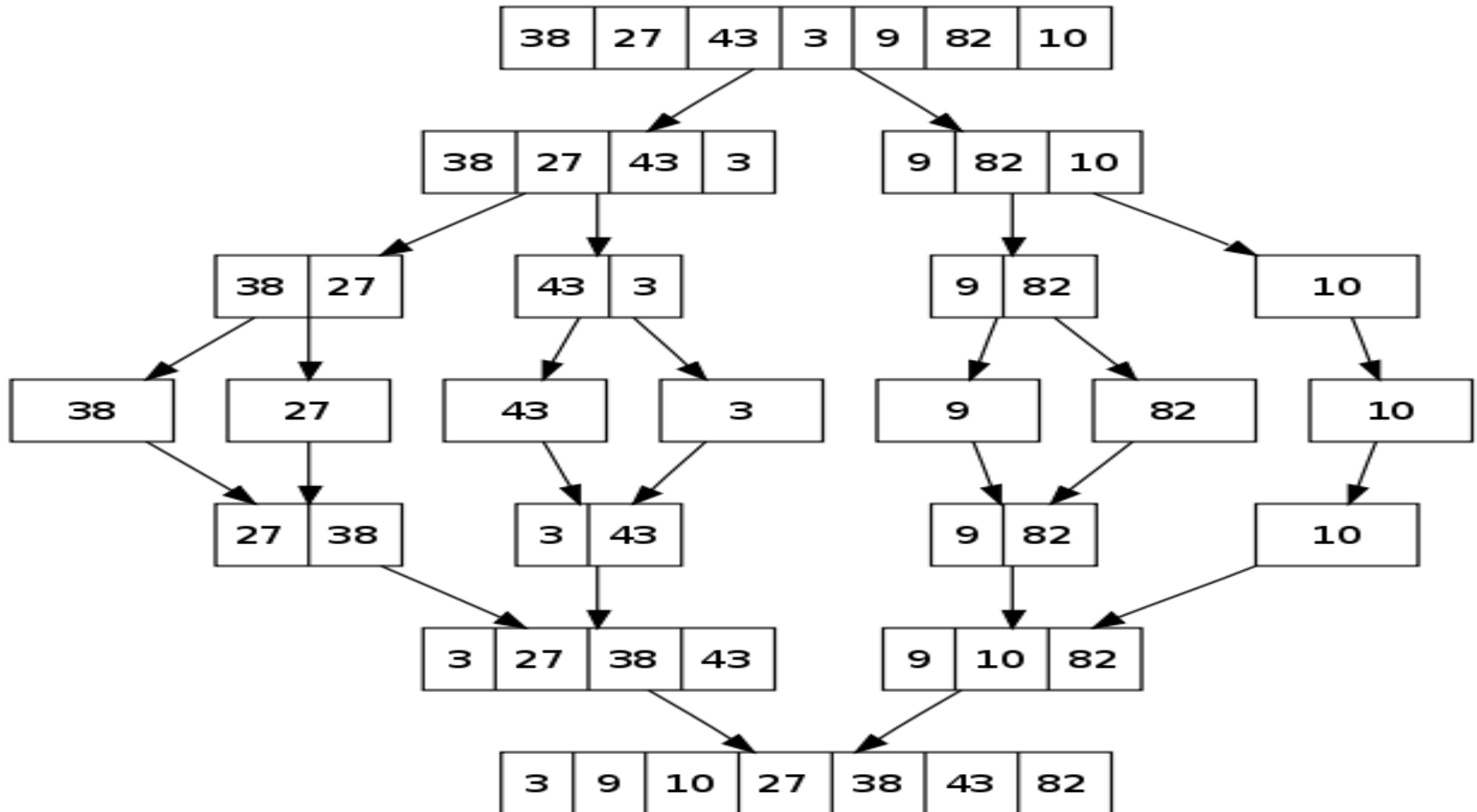
```
procedure bubbleSort( A : list of sortable items )  
  n = length(A)  
  repeat  
    newn = 0  
    for i = 1 to n-1 inclusive do  
      if A[i-1] > A[i] then  
        swap(A[i-1], A[i])  
        newn = i  
      end if  
    end for  
    n = newn  
  until n = 0  
end procedure
```

- Worst case and average case complexity is $\theta(n^2)$
- It is considered the worst sorting algorithm because it requires twice as many writes as insertion sort

- Divide and conquer algorithm
- Main idea:
 - Divide step:
 - Divide the array in two approximately equal pieces
 - Conquer step:
 - Recursively solve the two subproblems
 - Combine step:
 - Merge the two sorted subarrays into a single one

- Divide and conquer algorithm
- Main idea:
 - Divide step:
 - Divide the array in two approximately equal pieces
 - Conquer step:
 - Recursively solve the two subproblems
 - Combine step:
 - Merge the two sorted subarrays into a single one

Merge sort





Pseudocode:

– Merge function:

MERGE (A, p, q, r)

1. $n1 \leftarrow q - p + 1$
2. $n2 \leftarrow r - q$
3. Create arrays $L[1 \dots n1 + 1]$ and $R[1 \dots n2 + 1]$
4. FOR $i \leftarrow 1$ TO $n1$
5. DO $L[i] \leftarrow A[p + i - 1]$
6. FOR $j \leftarrow 1$ TO $n2$
7. DO $R[j] \leftarrow A[q + j]$

Merge sort



```
8.  L[n1 + 1] ← ∞  
9.  R[n2 + 1] ← ∞  
10. i ← 1  
11. j ← 1  
12. FOR k ← p TO r  
13.   DO IF L[i] ≤ R[j]  
14.     THEN A[k] ← L[i]  
15.       i ← i + 1  
16.     ELSE A[k] ← R[j]  
17.       j ← j + 1
```

- Time complexity
 - $T(n) = 2T(n/2) + \theta(n)$
 - Using case 2 of the master theorem merge sort has time complexity $\theta(n \log n)$
 - The main disadvantage of mergesort is that it uses extra memory in order to do the merge
 - Another disadvantage might be the recursion but it can be eliminated by using a bottom up approach

- An advantage is that it is a stable sort

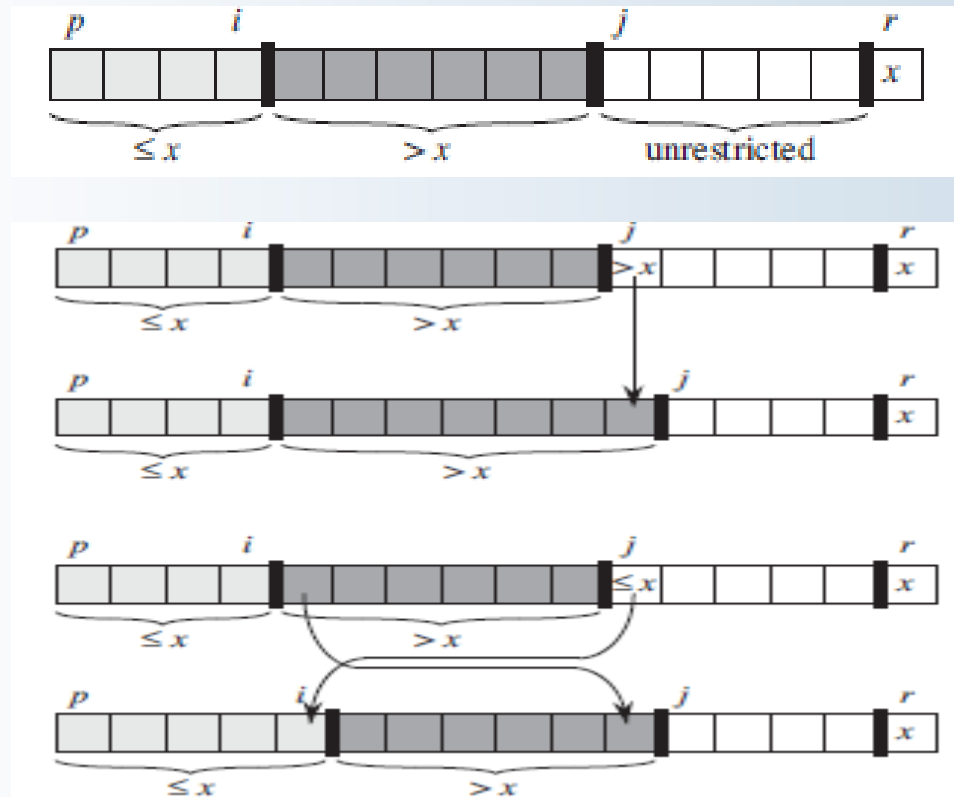
- Another divide and conquer algorithm
- Very elegant
- Was invented by David Hoare
- The three steps are:
 - Divide step:
 - In the divide phase an element from the array called the pivot is chosen
 - Using the pivot the array is divided in two subarrays

- In the left subarray will lie the elements that are less than or equal to the pivot
 - In the right subarray we will have the elements that are greater than the pivot
- Conquer step
- Solve in a recursive manner the two subarrays

- Combine step:
 - Nothing left to do here because when we return from the two recursive calls the two subarrays will be already sorted and the pivot is always in the correct position !

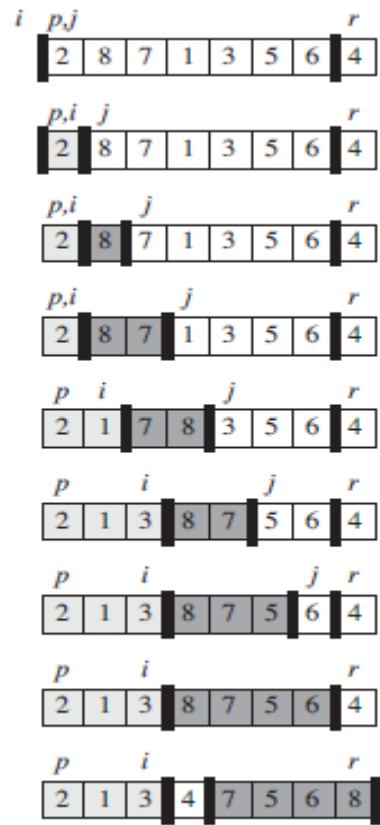
Quicksort

- The partition function maintains the following invariant:



Quicksort

Example of the partition function in action:





Partition function pseudocode:

- Function partition(A,left,right)
 - pivot=A[right]
 - i=left-1
 - for(j=left;j<right;j++)
 - if(A[j]<=pivot)
 - » i++
 - » swap(A[j],A[i])
 - swap(A[right],A[i+1])
 - return i+1

• The quicksort pseudocode is straightforward:

- Function Quick(A,left,right)
 - if(left<right)
 - pivot=Partition(A,left,right)
 - Quick(A,left,pivot-1)
 - Quick(A,pivot+1,right)

- Time complexity analysis
- The running time depends on how balanced the two partitions are
- This depends on how well we choose the pivot
- For example, if we choose the median the time complexity will be $\theta(n \log n)$
- The worst case behavior occurs when in all the recursive calls the pivot is the maximum or the minimum of the respective subarray
- In this case the time complexity will be $\theta(n^2)$

- The good news is that in the average case the time complexity using the partition function presented in slide 29 will be $\theta(n \log n)$
- The intuition is the following:
 - Any split of constant proportionality will yield a running time of $\theta(n \log n)$
 - It is assumed that all the permutations of the input numbers are equally likely
 - In the average case the partition function will produce a mix of good and bad partitions

- The problem is what if the permutations are not equally likely
- The solution used in practice is to choose the pivot randomly
- The modifications to the partition function are minimal
- Just choose a random element from the array and swap it with the last element
- In this case it is proved that the expected running time of quicksort will be $\theta(n \log n)$ if the elements of the array are distinct
- An even better method is to choose three random elements from the array and choose the pivot as the middle element of those three elements

Optimizing merge sort using insertion sort on small chunks



- Insertion sort on small vectors is in practice faster than other sorting algorithms
- We can do the following optimization:
 - Break up the array in n/k subarrays
 - Sort each subarray of size k using insertion sort
 - For n/k subarrays this will run in $\Theta(nk)$ worst case time
 - Then we will merge the n/k subarrays using the merge subroutine
 - This can be done in $\Theta(n \log(n/k))$ time

Optimizing merge sort using insertion sort on small chunks





- Thus, the total running time of the algorithm will be $\Theta(nk + n \log(n/k))$
worst case time
- How to choose k in practice ?

Optimizing merge sort using insertion sort on small chunks



- From tests I observed that for a chunk size of 100 insertion sort clearly outperforms mergesort
- The test was like this:
 - In a loop from 1 to 100000 I called mergesort for a vector that contained 100 random elements
 - The time for this was 12.48 seconds
 - I did the same thing using insertion sort
 - The time was 3.42 seconds !!!

-  Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, Third Edition
-  Wikipedia.org