UNIT 7.
SEARCH, SORT AND MERGE ALGORITHMS

Programming

Year 2017-2018

Industrial Technology Engineering

Paula de Toledo

Universidad Carlos III de Madrid
www.uc3m.es

CONTENTS

SEARCH

Search, sort and merge algorithms

- Search (search a value in a list)
  - in sorted list.
  - in unsorted list.
- Sort (a list of values)
  - Bubble sort algorithm.
  - Insertion sort algorithm.
  - Selection sort algorithm.
- Merge (two lists of values)
  - Merge two ordered lists into another

- Lists are represented as vectors
- In-place algorithms (use no extra memory space)

Search algorithms

- Search = find the position of a given value in a list

- Search in sorted lists
    - Linear search (or simple sequential search)
        - Examine each element starting from the firust until we find the value sought or the end of the list is reached

Linear search (sequential search) algorithm

```
int a[N] // vector containing the list
scanf("%i",&valuetosearch ); //value to search
//linear search
    i = 0 ;
    found = 0 ; // flag/marcador: values true/cierto(1)
               //                 false/falso(0)
    while((i<N)&&(!found)){
        if (a[i]==valuetosearch )
            found = 1 ;
        else
            i++;
    }
 // results - post check
 if (found){ //if a[i]==valuetosearch
    printf ("Value to search  found in position");
    printf ("%i", i+1);
 }
 else
    printf ("%s", " Value not found");
```

Search in sorted lists

- Can be optimized
  - More efficient (as in a dictionary)
  - ... although requires previous sort
- Two algorithms
  - **Optimized linear search**
    - Search ends when the element is found or the search goes beyond the position in the list where the value should be found
  - **Binary search**
    - check the central element of the list.
      - If it's the element sought the search ends
      - If not, repeat using only the part of the list where the element should be
        - Until the sub-list is empty

Optimized sequential (lineal) search

```
int list[N]
i = 0 ;
found = 0 ; // flag to control if the value is found 0 (false) 1 (true)
end = 0;   // flag to control search end 0 (false) 1 (true)

scanf("%i",&valuetosearch );
while((!end) && (!found)){
    if (list[i]==valuetosearch )
        found = 1 ;
    else {
        if ( (list[i]>valuetosearch ) || (i==(N-1)) )
          // i beyond position where value shoud be or i= end of the list
          end= 1 ; // end search
        else
          i++;
    }
 }
// results - post check
 if (found){ //if a[i]==valuetosearch
    printf ("Value to search  found in position "%i", i+1);
 }
 else
    printf ("%s", " Value to search not found");
```

Binary search - concept

valuetosearch = 37

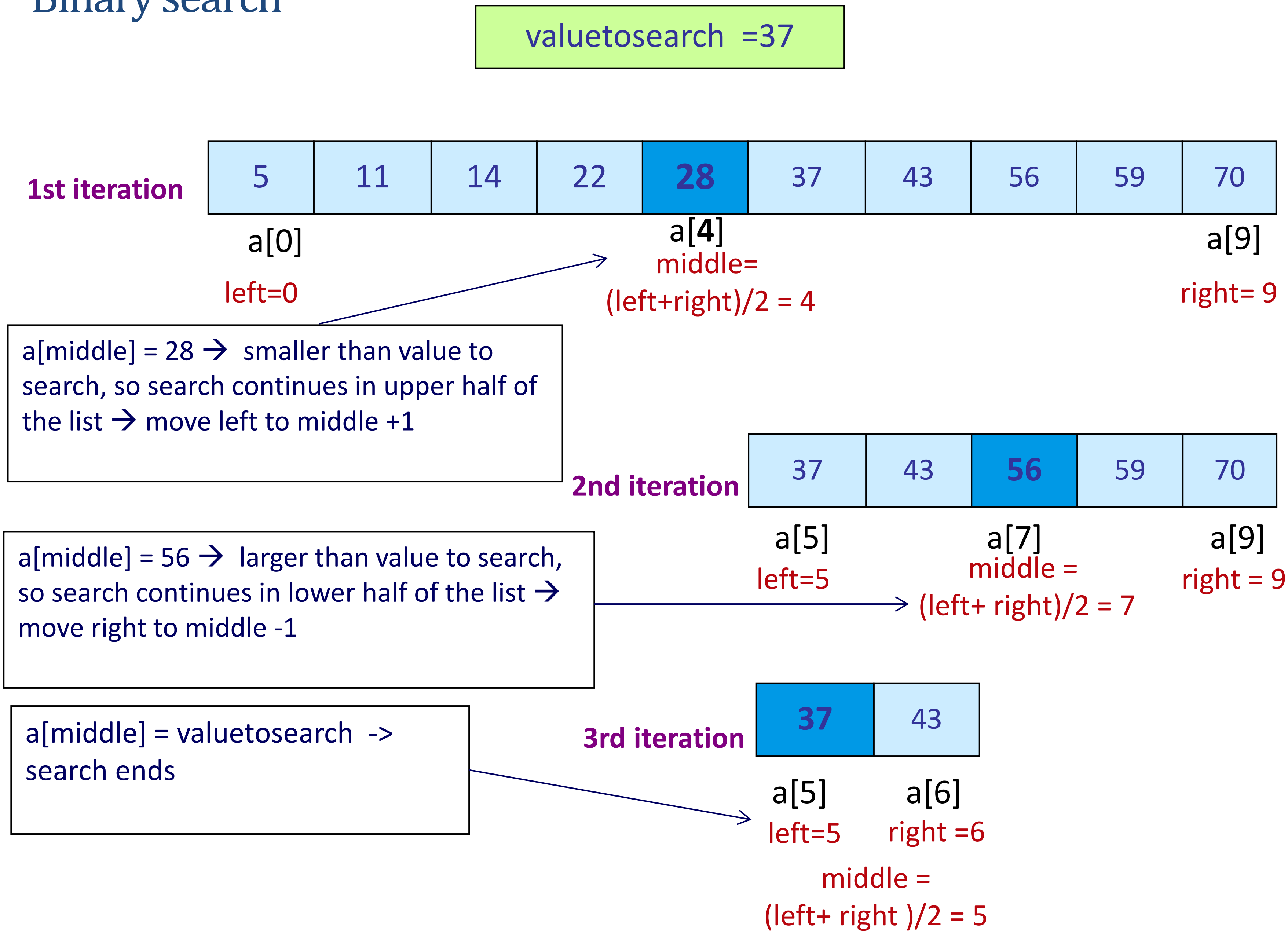
5	11	14	22	28	37	43	56	59	70

37	43	56	59	70

37	43

valuetosearch = 38

Binary search



Binary search

valuetosearch =7
(not in the list)

1st iteration

a[0]				a[4]					a[9]
5	11	14	22	**28**	37	43	56	59	70

left=0
middle= (left+ right)/2=4
right= 9

a[middle] = 28 → larger than value to search, so search continues in lower half of the list

2nd iteration

a[0]	a[1]		a[3]
5	**11**	14	22

left=0
middle = (izq+ right)/2 = 1
right =3

a[middle] = 11 → larger than value to search, so search continues in lower half of the list → move right to middle -1

3rd iteration

a[0]
5

middle = (left+ right)/2 = 0

left=0 right =0

a[middle] = 5 → smaller than value to search, so search continues in upper half of the list → move left to middle + 1

Iteración 4:

left=1 right= 0
→ empty list

4 –left > right → the value is not in the list, search ends

Binary search

```
# define N 10    //vector size
int a[N];        //vector
int left, right, middle; // indexes for leftmost and righthmost
                           elements in the sublist
int valuetosearch ;
int found;    //flag true(1) or false(0)
int position=-1;      //position of the sought value in the vector,
                       intialized to -1
//initialization
  left = 0;  // lower limit (left) of sublist
  right = N-1;  //upper limit (right) of sublist
  found = 0;
  middle=(left+right)/2;
```

Binary search

```
   scanf("%i",&valuetosearch );

   while ((left<=right) && (!found)){
       if (a[middle] == valuetosearch ){
              found=1;
              position= middle;
      } else {
             //if a[middle] is smaller, move left to m+1
             if (a[middle] < valuetosearch )
                 left = middle + 1;
             //if a[middle] is larger, move the left to m-1
             else
                  right = middle - 1;
         // get new middle
         middle = (left+ right)/2;
       } // if
    } // while
```

Binary search

```
  // post check and results

  if (found)   {
     printf ("value %i found in position %i" ,valuetosearch,
position);
       }else{
         printf (value %i not found", valuetosearch );
       }
```

SORT

Sorting algorithms

- Basic sorting algorithms (http://www.sorting-algorithms.com/)

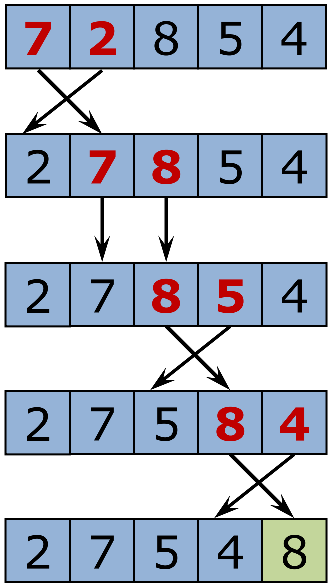
  - **Exchange (Bubble) sort:**
    - Compare adjacent values and **swap** them until list is ordered
  - **Direct selection sort**
    - **Select** the smaller value of the list and move it to the first position; then select second smallest element and put it in second position, …. Until the last element is sorted
  - **Direct insertion sort**
    - Work with a sublist that is sorted, and **insert** the remaining elements in the corresponding position of the list. Initially the orered sublist is only one element, and it grows until all elements are sorted

Bubble algorithm

- Example: four elements, ascending order
  - First iteration:
    - Compare each element (except the last) to the element after it
    - If order is wrong, sort them
      - If m(j) > m (j+1), swap
    - The larger element will move to the end, it's sorted
  - Second iteration:
    - Compare each element (except the two last) to the element after it
    - If order is wrong, swap
    - The second larger moves to the penultimate postion
    - Two last elements are sorted
  - Third iteration:
    - Compare each element (except the three last) to the element after it
    - Three last elements are sorted, therefore full list is sorted

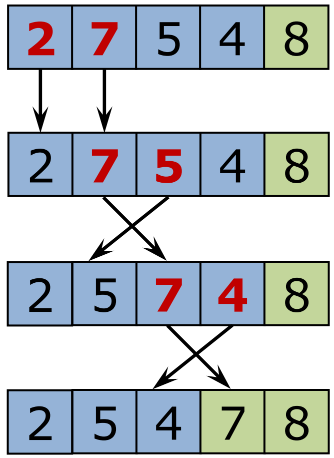
Bubble sort

Sort vector a[] of **five** elements (N = 5) in **increasing order**

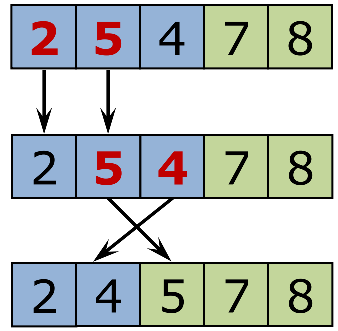


1st iteration:
(i=1)
The largest element is sorted

4 comparisons
N-i comparisons

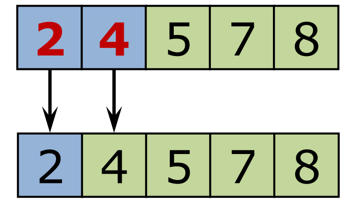


2nd iteration :
(i=2)
The two largest elements are sorted
3 comparisons
N-i comparisons



3rd iteration :
(i=3)
The three largest elements are sorted

2 comparisons
N-i comparisons



(end)

4th iteration:
(i=4)
The four largest elements are sorted
-> all are sorted
1 comparison
(N-i) comparisons

You need 4 (N-1) iterations=

Every round you need less comparisons, starting from N-1, down to 1

Bubble sort - code

```
For (i=1; i<=N-1; i++) { // N-1 iterations
      for(j=0; j < N-i; j++) { // for N-i elements, compare each
                                 element to the next
      // if they are not ordered, swap using auxiliary variable
      if(a[j]>a[j+1]) {
          aux=a[j];
          a[j]=a[j+1];
          a[j+1]=aux;
       }
   }
}
```

Selection sort algorithm

- Select the smallest element, put it in the first position
- Among the remaining elements find the smallest and put it second
  - Sorted list grows, unsorted list shrinks
- Continue until they are all sorted

- Given an N – element vector
  - Find the smallest in the range 0 to N-1
    - Exchange (swap) it with the element in position 0
  - Find the smallest in the range 1 to N-1
    - Swap it with the element in position 1
  - Find the smallest in the range 2 to N-1
    - Swap it with the element in position 2
  - Find the smallest in the range 3 to N-1
    - Swap it with the element in position 3
  - Continue until sorted list size is N

Selection sort

Example, N=5

i denotes the first element in the unsorted list

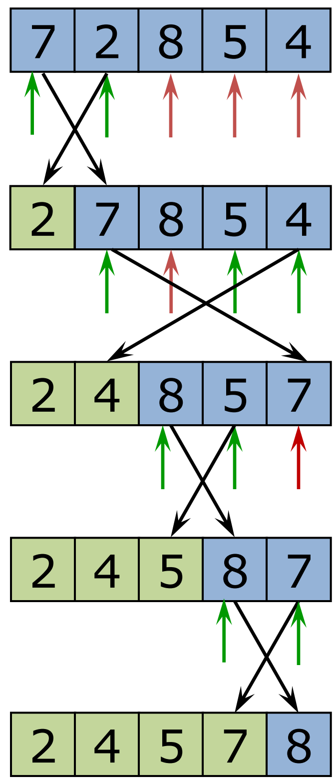
j is the index used to search the minimum in the sublist

j from 1 to N-1

j from 2 hasta N-1

j from 3 hasta N-1

j from 4 hasta N-1



First iteration
First element in unsorted list i=0;
posMin =1;

Second iteration,
First element in unsorted list i=1;
posMin =4;

Third iteration,
First element in unsorted list i=2;
posMin =3;

Fourth iteration
i=3;
posMin =4

END

4 (N-1)
Iterations needed

Selection sort - code

```
for(i=0;i<N-1;i++){ //N-1 iterations
    // find the minimum in the sublist and it's position
    min = a[i]; // initialize min to first element in sublist
    posMin = i; // initialize  position
   // search in the unsorted list, starting in i to N_1
    for(j=i; j<=N-1; j++){
        if (a[j] < min) {
              min = a[j];
              posMin= j;
        }
    }
    //swap the first element of the unsorted list a[i]
    // with the minimum
    a[posMin] = a[i];
    a[i] = min;
}
```

min stores the smallest value in the sublist

posMin is the index of the smallest value in the sublist

Insertion sort

- Start with an ordered sublist, take next element and put it in its position in the list
- The sorted sublist grows, and at the end the list is sorted
-
- Starting point is a sorted sublist with only one element, the first one in the list

Insertion sort

Example, N=5

Ejemplo con N=5

i is the element to put in place

7	2	8	5	4

First iteration =1
Sorted sublist has one element
Work with the first element of the unsorted list a[1]=2
Move it until the sublist is sorted

2	7	8	5	4

Second iteration i=2
Sorted sublist has two elements
the first element of the unsorted list a[2]=8

2	7	8	5	4

Third iteration i=3
Sorted sublist has three elements
the first element of the unsorted list a[3]=5

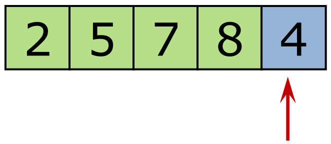
2	5	7	8	4

Fourth iteration i=4
Sorted sublist has four elements
the first element of the unsorted list a[4]=4

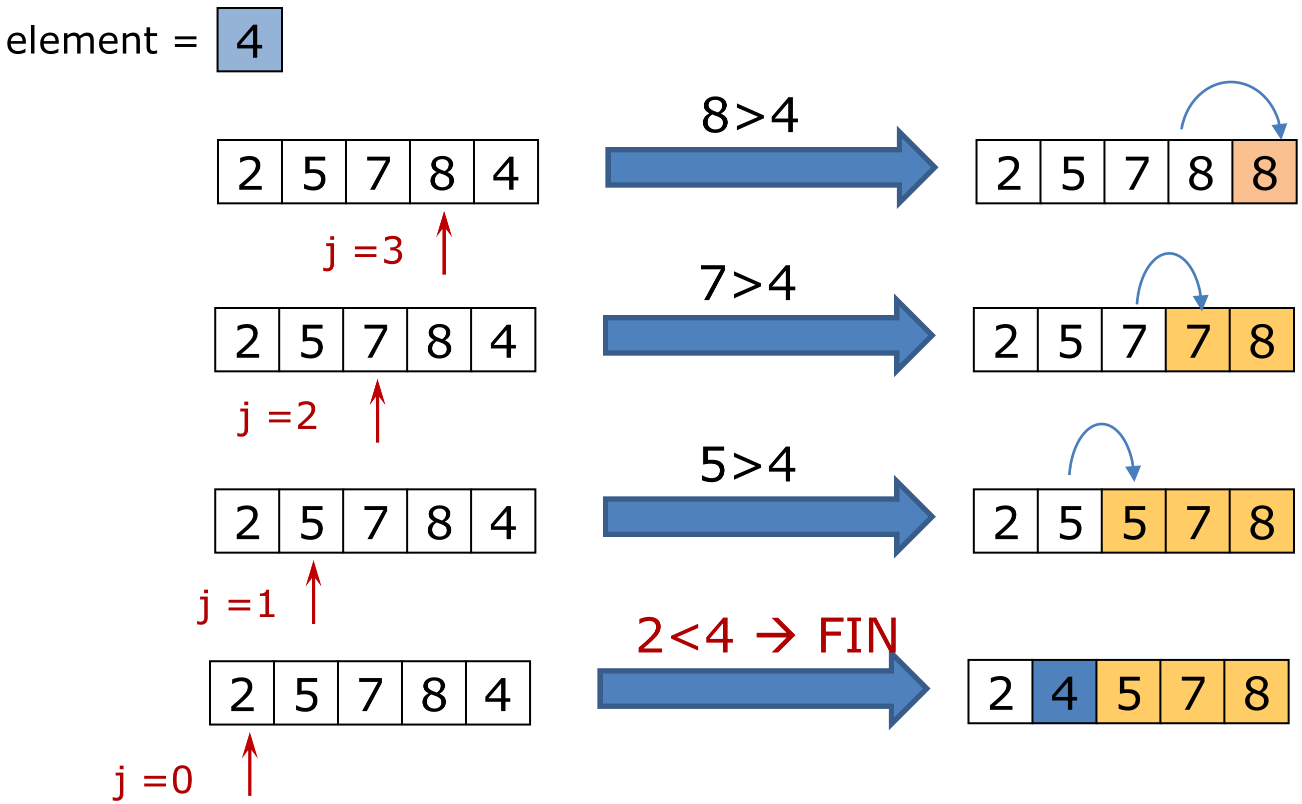
2	4	5	7	8

END

Insertion sort – shifting values to position new element



Fourth iteration (i=4),
The ordered sublist contains 4 elements
The element a[4]=4 has to be inserted in place



All the elements higher than the a[4] =4 are shifted to the right.

END

The element is placed at position j

Insertion sort - code

```
int a[N] = {7,2,8,5,4}; //vector N elements
int i,j, valuetosearch ; //i is the iteration
                         //j the index to use in the unsorted list
for (i=1; i<N; i++){
  // for each iteration i, the sorted sublist has i elements
  // we need to sort the i-th element, valuetosearch
  valuetosearch =a[i];
  // shift all elements of the sorted sublist to "make room" for
valuetosearch
  // starting from the end of the sorted list i-1
  j=i-1;
  //keep shifting elements while the position is not reached
  // a[j]>valuetosearch and the beginning of the list is not found
(j>0)
 while ((j>=0) && (a[j]>valuetosearch ) ){
      a[j+1]=a[j];
      j= j-1;
  }
  // put value to search in place
  a[j+1]=valuetosearch ;
}
```

Comparing sorting algorithms

- How long does it take to sort an array of n elements?
  - **Algoritm performance** is measured according to the number of comparisons and swapping operations that are required to obtain the solution
    - Performance strongly depends on the starting situation: sorted array, unsorted, nearly sorted, reversed, etc.
      - What algorithm will profit from a nearly sorted start?
  - In general, basic sorting algorithms (bubble, selection, insertion) are not suitable for large lists
    - the differences for small arrays are not significant
  - There are more efficient (and complex) sorting algorithms: shell, heap, merge, quicksort, etc.

Shell sort

- Shell = insertion in decreasing increments
  - Generalization of insertion sort where the elements to insert are not next to each other
  - The method starts by sorting pairs of elements far apart from each other, then progressively reducing the gap between elements to be compared
  - First iteration gap is half the size of the vector
    - If N=100 gap= 50
  - Last pass gap is 1
  - Every pass gap decreases (25, 12, 6, 3, 1)
  - Example
    - http://www.youtube.com/watch?v=QTtHQVRiD04

Quicksort

- Quick Sort (http://www.youtube.com/watch?v=cNB5JCG3vts)
  - Pick an element, called a pivot, from the array
  - Partitioning: reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position.
  - Recursively apply the above steps to the sub-array of elements with smaller values and separately to the sub-array of elements with greater values
    - Stop where size of the group is zero or one
    - If size is two just compare elements and swap if needed

Quicksort. example

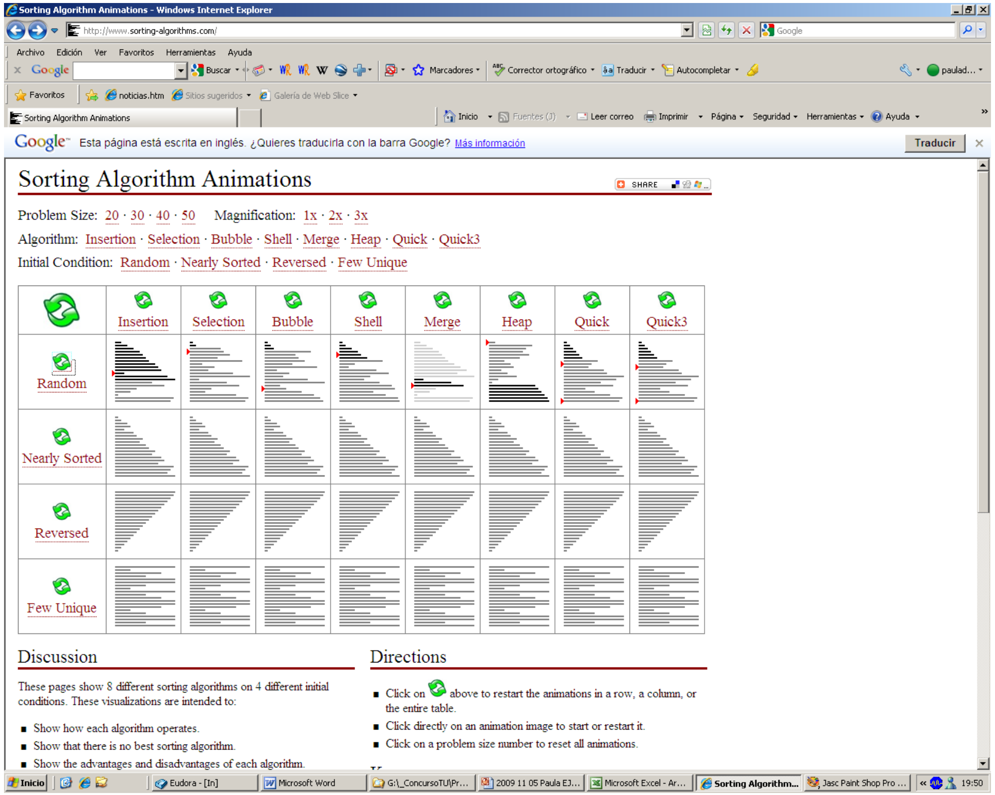
12	6	15	24	7	31	21	3	19	8
6	7	3	8	**12**	**15**	24	31	21	19
3	**6**	**7**	8	**12**	**15**	**24**	31	21	19
3	**6**	**7**	**8**	**12**	**15**	21	19	**24**	31
3	**6**	**7**	**8**	**12**	**15**	**19**	**21**	**24**	**31**

» http://en.wikipedia.org/wiki/Quicksort

- Is the number of comparisons and swaps much smaller than in bubble sort?

Comparing sorting algorithms

- http://www.sorting-algorithms.com/



Source: http://www.sorting-algorithms.com/ [link]

MERGE

Merging

- Merging
  - combining sorted sequences of values into a single sorted sequence
- Some advanced sorting algorithms use merging
  - The list is divided in smaller pieces to be ordered and, finally, the parts are merged
    - E.g.: Mergesort
  - 'Divide & Conquer' strategy: a problem can be solved by splitting it into parts, solving the parts, and joining the partial solutions
- Merging is also necessary if the number of values to sort is larger than the memory size
  - Divide in sublist, sort, and merge

Merging

- Idea:
  - Define two indices i,j to traverse all the elements of list1,list2 respectively
  - Determine the value to add to the merged array by comparing list1[i] and list2[j]
  - Copy the remaining elements of the list that has not been completely traversed to list

- **Input:**
  - `list1 ← {2, 4, 5, 7, 8}`
  - `T1 ← 5`
  - `list2 ← {1, 3, 8}`
  - `T2 ← 3`
- **Output:**
  - `list ← {1, 2, 3, 4, 5, 7, 8, 8}`

merge

```
const int T1=100, T2=50; // list size
int list1[T1],list2[T2],list3[T1+T2];
//two sorted lists (list1 and list2) and a final list
int i1,i2,i, k; //indexes
i1=0; i2=0; i=0;

while((i1<T1)&&(i2<T2)){
      if (list1[i1]<list2[i2]){
             list3[i]= list1[i1];
             i1 =i1+1;
   }
      else{
             list3[i]= list2[i2];
             i2=i2+1;
   }
      i=i+1;
}
…
```

merge

```
...
//final part of the longest list is pending
if (i1<T1)
// list 1 was longer, copy the remainder
      for (k=i1; k<T1; k++){
             list3[i] = list1[k];
             i=i+1;
      }
else
// list 2 was longer, copy the remainder
      for (k=i2; k<T2; k++){
             list3[i] = list2[k];
             i=i+1;
      }
```

UNIT 7.
SEARCH, SORT AND MERGE ALGORITHMS

Universidad
Carlos III de Madrid
www.uc3m.es