Data Structures

Sorting methods for arrays

Ingrid Kirschning

UDLAP

Some of the best known sorting methods for arrays

- Selection sort.
- Bubble sort.
- Merge sort.
- Quicksort.
- Radix sort.

UDLAP

Sorting means:

- To move the data or their references
- in order to create a sequence that represents a certain order.
- This order can be
  - Numerical ,
  - alphabetic or
  - alphanumeric,
  - ascending or
  - descending.

¿Why sort data?

- Data structures are used store data and information, so
- To be able to recover it efficiently it should be ordered.
- There are various methods to sort different basic data structures.
- Generally the sorting algorithms are not used frequently, in some cases only once.

Advantages and Disadvantages

- There are some simple sorting methods very easy to implement, useful when the number of elements to sort is not too large (for example 500 elements).

- There are sophiticated methods, more complex but more efficient in their execution time.

Simple sorting methods

- The simple sorting methods require aproximately n x n steps to sort n elements.
- These methods are:
  - insertion sort (or direct insertion)
  - selection sort,
  - bubble sort, and
  - shellsort, which is an extension of the insertion sort, but faster.

Complex sorting methods

- The more complex sorting methods are:
  - quicksort,
  - heap sort,
  - radix sort and
  - address-calculation sort.

UDLAP

Notación O

- The efficiency of the algorithms can be measured in different ways. They are usually based on the number of comparisons and data movements the algorithm performs.
- The best, worst and average cases are generally analyzed.
- An algorithm performs **O(n^2)** comparisons when it compares n times the n elements, n x n = n^2, where n is the number of elements in the array.

UDLAP

Analysis of Algorithms: "Big O" Notation

If $T(n)$ represents the execution time of an algorithm and $f(n)$ is some expression for it's upper limit, $T(n)$ is contained in the set $O(f(n))$, if there are two positive constants c and n_0 such that

$$|T(n)| \leq c\,|f(n)| \text{ for each } n > n_0$$

Example:

$100\ n^3 \Rightarrow O(n^3)$

$6n^2 + 2n + 4 \Rightarrow O(n^2)$

$1024 \Rightarrow O(1)$

$1+2+3+4+\ldots+ n-1 + n = n * (n+1)/2 = O(n^2)$

UDLAP

Insertion Sort

- This is one of the simplest methods.
- It consists in taking one by one the elements of the array (starting at the Step 2nd position) and compare it to the 1st element. Moving it if they are not sorted correctly.
- Next it takes the 3rd element and compares it to the 2nd and the 1st.
- The result is that the array gets sorted incrementally from the first position to the last.

UDLAP

Insertion Sort

This algorithm works on the array to be sorted called a[] and it modifies the positions of its elements until they are sorted from the smallest to the largest. N represents the number of elements in the array a[]. (The algorithm assumes that the first position in the array is the #1)

Step 1: [i starts at the second position]	For i = 2 to N do
Step 2: [v stores the content of pos. I, and j stores the value of i]	Set v = a[i], j = i.
Step 3: [Compare v to the data before]	While a[j-1] > v AND j>1 do
Step 4: [if v is smaller values are moved forward]	set a[j] = a[j-1],
Step 5: [j continues backwards]	set j = j-1.
Step 6: [after moving larger values, v is inserted into its correct position]	Set a[j] = v
Step 6: [End]	end for

UDLAP

Selection Sort

- The selection sort method consists in finding the **smallest** value of all in the array and exchanging it with whatever is in the first position of the array.
- Next it searches for the second smallest value in the array and exchanges it with the one in the second position in the array.
- This process is repeated for all the elements in the array.

UDLAP

Selection Sort

Step 1: [i – for every value in the array]

Step 2: [initialize the position of the smallest value]

Step 3: [j traverses the rest of the array]

Step 4: [if position j has a smaller value tan position min]

Step 5: [min is re-assigned to point to j]

Step 6: [After comparing all, the values in position i and min are exchanged]

Step 7: [End]

```
For i = 1 to N do
    min= i

    For j = i+1 to N do
        If a[j] < a[min] then
            min = j
    Swap(a, min, i).
End.
```

(The algorithm assumes that the first position of the array is the position #1, N is the total number of items in the array, and 'a' is the name of the array)

UDLAP

Algorithm complexity

The number of comparisons this algorithm performs is:

- To sort the **1st** element **n-1** comparisons are made
- For the **ith** element **n-i** comparisons are made;
- And for n elements in the array this process is repreated n-1 times, if we add them the result is:

$$T(n) = (n-1)+(n-2)+(n-3)+\ldots+2+1$$

Which is an arithmetic series and can be re-written as:

$T(n) = n(n-1)/2$

And with this we can conclude that the complexity is $O(n^2)$

UDLAP

Bubblesort

- The algorithm works the following way:
- It traverses the array comparing the data pair by pair and swapping those that are not ordered.
- This is repeated as many times as there are still changes (swaps).
- What happens is that during the fist time it traverses the array the largest value is moved to the last position. During the next run it will move the second-largest, and so on until all are sorted.

UDLAP

Bubblesort

Step 1: [i starts at the end of the array]	For i = N downto 1 do
Step 2: [j starts at the second item]	For j = 2 to i do
Step 3: [If the 2 values are not in order]	If a[j-1] < a[j] then
Step 5: [Exchange both values]	Swap(a, j-1, j).
Step 6: [End]	end

(The algorithm assumes that the first position of the array is the position #1, **N** is the total number of items in the array, and ‘**a**’ is the name of the array)

UDLAP

¿Cuáles son los tiempos de ejecución?

- Review the algorithm for Bubblesort and analyze the execution time for the best case, the worst case and the average to determine the complexity using the "Big O" notation.

Merge Sort

- Quicksort divides the array in two and sorts each half recursively.
- MergeSort Works the opposite way: here the method joins two sorted structures to create one new sorted structure.
- Advantage: it uses an execution time proportional to:

  n log (n),
- Disadvantage: the method requires extra space to perform the procedure.
- This type of method is useful when you already have one ordered structure and the new data to be added are stored in another temporal structure to be merged later.

UDLAP

MergeSort

```
/*receives 2 indexes: l for the lower limit of the array to be sorted and r for the upper
limit*/
void mergesort    (int l, int r)
{     int  i,j,k,m,b[MAX]; if (r > l)
              {
                          m = (r+l) /2;
                          mergesort (l, m);
                          mergesort (m+1, r);
                          for (i= m+1; i > l ; i--)
                                    b[i-1] = arr[i-1];
                          for (j= m; j < r; j++)
                                    b[r+m-j] = arr[j+1];
                          for (k=l ; k <=r; k++)
                                    if(b[i] < b[j])
                                              arr[k] = b[i++];
                                    else
                                              arr[k] = b[j--];
              }
     }
```

UDLAP

Quicksort

- This method is also known as Partition-Exchange Sort.
- It is a recursive sorting method
- In programming languages where no recursión is posible the execution time can be significantly slower.
- The average execution time for quicksort is

  $n \log_2 n$

```
void quicksort(int a[], int l, int r)
{           int i,  j,  v;
            if(r > l)
            {           v = a[r];
                        i = l-1;
                        j = r;
                        for(;;)
                        {           while(a[++i] < v && i <
                                    r); while(a[--j] > v && j >
                                    l); if( i >= j)
                                                break;
                                    swap(a,i,j);
                        }
                        swap(a,i,r);
                        quicksort(a,l,i-1);
                        quicksort(a,i+1,r);
            }
}
```

Radix Sort

- This method sorts data by their components.
- An integer is decomposed into units,
- tens, hundreds, thousands … and
- Sorts all the numbers first according to the value of their units
- Then it re-orders the numbers by the value of their tens, then hundreds, then thousands and so on until all numbers are sorted by thier most significant digit (leftmost)

UDLAP

```
public void sort()
    {  int i, m = a[0], exp = 1, n = a.length;
       int[] b = new int[10];
       int[] bucket = new int[10];  // WE ASSUME THEY ARE CREATED FULL OF ZERO'S
          for (i = 1; i < n; i++)  // SEARCHES FOR THE LARGEST VALUE IN THE ARRAY
              if (a[i] > m)
                  m = a[i];
          while (m / exp > 0) {     // BEGINS SORTING
                for (i = 0; i < n; i++) {  // BUCKET COUNTS THE NUMBER OF REPEATED DIGITS
                         k = a[i] / exp) % 10 ;
                         bucket[k] = bucket [k]+1;
                 }
                for (i = 1; i < 10; i++)  // ADJUSTMENT OF BUCKET
                         bucket[i] = bucket[i]  + bucket[i - 1];
                 for (i = n - 1; i >= 0; i--) { // SORTS INTO b THE ELEMENTS OF a
                         p = a[i] / exp) % 10;
                         bucket[k] = bucket[k] -1;
                         s = bucket [k];
                         b[s] = a[i];
                 }
                 for (i = 0; i < n; i++)
                         a[i] = b[i]; // REPLACES THE NEWLY SORTED ELEMENTS IN a
               exp =  exp * 10;  // NOW THE NEXT SIGNIFICANT DIGIT
          } // END WHILE
       }        // http://www.sanfoundry.com/java-program-implement-radix-sort/
```

Example

Data to de sorted:	1st by the least significant digit:	2nd by the following digit:	3rd by the following digit:
34	841	06	006
6	62	123	034
237	123	34	062
123	34	237	123
62	6	841	237
841	237	62	841

UDLAP

Summary of execution times

Algorithm	Worst-Case running time	Average/Expected running time
Insertion sort	$O(n^2)$	$O(n^2)$
Merge sort	$O(n \log n)$	$O(n \log n)$
Heapsort	$O(n \log n)$	
Quicksort	$O(n^2)$	$O(n \log n)$
Bubble sort	$O(n^2)$	$O(n^2)$, and best case: $O(n)$
Radix sort	$O(d(n+k))$ d: # digits, k:possible values of each digit	$O(n)$ for constant d and k = $O(n)$
Shellsort		$O(n^{1.25})$

UDLAP