

Data Structure

Project #4

Types of Sorting

### Mohammad Hamdan

1111789

Selection Sort :

Selection sort Algorithem depend on finding the smallest or largest (depend on you ) integer at the array , then exchange the thes integer with leftmost and moving to right .

Example :

14 52 1 78 45

1 52 14 78 45

1 14 52 78 45

1 14 45 78 52

1 14 45 52 78

Code :

int i,j;

int iMin;

 for (j = 0; j < n-1; j++) {

 /\* find the min element in the unsorted a[j .. n-1] \*/

 /\* assume the min is the first element \*/

 iMin = j;

 /\* test against elements after j to find the smallest \*/

 for ( i = j+1; i < n; i++) {

 /\* if this element is less, then it is the new minimum \*/

 if (a[i] < a[iMin]) {

 /\* found new minimum; remember its index \*/

 iMin = i;

 }

 }

 /\* iMin is the index of the minimum element. Swap it wit the current position \*/

 if ( iMin != j ) {

 swap(a[j], a[iMin]);

 }

}

Time :

Worst Case :- [O](http://en.wikipedia.org/wiki/Big_O_notation)(*n*2)

Best Case :- O(n) [1]

Bucket Sort :

Bucked Sort Algorithem is to divide the array into bucket’s , each bucket deal with it as individually . and match the integer in array to right bucket . It take Ω(n log n) .

|  |  |
| --- | --- |
| [**Worst case performance**](https://en.wikipedia.org/wiki/Best%2C_worst_and_average_case) | O(n^2) |
| [**Average case performance**](https://en.wikipedia.org/wiki/Best%2C_worst_and_average_case) | O(n+k) |
| [**Worst case space complexity**](https://en.wikipedia.org/wiki/Best%2C_worst_and_average_case) | O(n\cdot k) |



Code :

**function** bucketSort(array, n) **is**

 buckets ← new array of n empty lists **for** i=0 **to** (length(array)-1) **do**

 insert *array[i]* into buckets[msbits(array[i],k)]

**for** i = 0 **to** n - 1 **do**

 nextSort(buckets[i]);

**return** the concatenation of buckets[0], ...., buckets[n-1]

Advantages :the user knows the range of the elements;

time complexity is good compared to other algorithms.

Disadvantages : you are limited to having to know the greatest element extra memory is required . [2]

Cocktail sort :

Cocktail sort is a stable comparison sorting algorithm. It is a variation of bubble sort.

 In bubble sort, values only bubble in one direction. In cocktail sort, values bubble both directions, thus avoiding turtles.

Code :

**procedure** cocktailSort( A **:** list of sortable items ) **defined as:**

 **do**

 swapped := false

 **for each** i **in** 0 **to** length( A ) - 2 **do:**

 **if** A[ i ] > A[ i + 1 ] **then** // test whether the two elements are in the wrong order

 swap( A[ i ], A[ i + 1 ] ) // let the two elements change places

 swapped := true

 **end if**

 **end for**

 **if** swapped = false **then**

 // we can exit the outer loop here if no swaps occurred.

 **break do-while loop**

 **end if**

 swapped := false

 **for each** i **in** length( A ) - 2 **to** 0 **do:**

 **if** A[ i ] > A[ i + 1 ] **then**

 swap( A[ i ], A[ i + 1 ] )

 swapped := true

 **end if**

 **end for**

 **while** swapped // if no elements have been swapped, then the list is sorted

**end procedure**

Time:

Worst-case: O(n²)

Best-case: O(n) [3]

Strand sort :

Strand sort is a sorting algorithm. It works by repeatedly by pulling the first element in list , next element should larger then the first , when we finished we apply the same to the remaining item in list , merges these list together , we finiched when we have one number and that will be the smallest , we merge it at the first .

Code :

procedure strandSort( A : list of sortable items ) defined as:

 while length( A ) > 0

 clear sublist

 sublist[ 0 ] := A[ 0 ]

 remove A[ 0 ]

 for each i in 0 to length( A ) - 1 do:

 if A[ i ] > sublist[ last ] then

 append A[ i ] to sublist

 remove A[ i ]

 end if

 end for

 merge sublist into results

 end while

 return results

end procedure

Time :

Worst-case :- O(n²)

Best-case :- O(n) [4]

Gnome sort :

In this sort the Algorithem is comparing the first element with the next element , the first should be smaller than the next if not we swap , on each swaping we cheak backword if there are any change happened then continue until the last element .

Code :

procedure gnomeSort(a[])

 pos := 1

 while pos < length(a)

 if (a[pos] >= a[pos-1])

 pos := pos + 1

 else

 swap a[pos] and a[pos-1]

 if (pos > 1)

 pos := pos - 1

 end if

 end if

 end while

end procedure

Time :

Worest Case : 

Best Case : O (n)

Avarage Case :  [5]

References :

[1] : <http://en.wikipedia.org/wiki/Selection_sort>

[2] : <http://en.wikipedia.org/wiki/Bucket_sort>

[3] : <http://en.wikipedia.org/wiki/Cocktail_sort>

[4] : <http://en.wikipedia.org/wiki/Strand_sort>

[5] : <http://en.wikipedia.org/wiki/Gnome_sort>