## Bubble Sort

|  |  |
| --- | --- |
| init | 1 4 3 9 5 8 7 |
| 1 | 1 4 3 5 8 7 **9** |
| 2 | 1 4 3 5 7 **8** 9 |
| 3 | 1 3 4 5 **7** 8 9 |
| 4 | 1 3 4 **5** 7 8 9 |
| 5 | 1 3 **4** 5 7 8 9 |
| 6 | 1 **3** 4 5 7 8 9 |

### Algorithm

Each time bubble the largest element to the right side.

For array A[n]:

The first round bubble A[0 .. n-1], result in the max element bubbled to A[n-1]

The second round bubble A[0 .. n-2], result in the max element bubbled to A[n-2]

…

The last (n-1) round bubble A[0 .. 1], result in the max element bubbled to A[1]

### Pseudocode

Bubble Sort A[n]

For i = n-1 to 1 (total rounds: n-1)

For j = 0 to i

If (A[j] < A[j + 1])

Swap(A[j], A[j+1])

## Selection Sort

|  |  |
| --- | --- |
| init | 1 4 3 9 5 8 7 |
| 1 | **1** 4 3 9 5 8 7 |
| 2 | 1 **3** 4 9 5 8 7 |
| 3 | 1 3 **4** 9 5 8 7 |
| 4 | 1 3 4 **5** 9 8 7 |
| 5 | 1 3 4 5 **7** 9 8 |
| 6 | 1 3 4 5 7 **8** 9 |

### Algorithm

The first round mark index(0) as minimal, find index(i) that have smallest value in index(0..n-1), swap(0,i)

### Pseudocode

Selection Sort A[n]

for i = 0 to n-2

min = i

for j = i+1 to n-1

if (A[j] < A[min])

min = j

swap(A[i], A[min])

## Insertion Sort

|  |  |
| --- | --- |
| init | 1 4 3 9 5 8 7 |
| 1 | 1 **4** 3 9 5 8 7 |
| 2 | 1 **3** 4 9 5 8 7 |
| 3 | 1 3 4 **9** 5 8 7 |
| 4 | 1 3 4 **5** 9 8 7 |
| 5 | 1 3 4 5 **8** 9 7 |
| 6 | 1 3 4 5 **7** 8 9 |

### Algorithm

Each time put A[n] into sorted sequence A[0 .. n-1]

The first round put A[1] into sorted sequence A[0]

The second round put A[2] into sorted sequence A[0..1]

..

The last round put A[n-1] into sorted sequence A[0..n-2]

### Pseudocode

Insertion Sort A[n]

For i = 1 to n – 1

Key = A[i]

While i > 0 && key < A[i - 1]

A[i - 1] = key

i--

A[i] = key

## Shell Sort

[ 13 14 94 33 82 25 59 94 65 23 45 27 73 25 39 10 ] step 5,3,1

13 14 94 33 82

25 59 94 65 23

45 27 73 25 39

10

10 14 73

25 23 13

27 94 33

39 25 59

94 65 82

45

At last insertion sort

10 14 13

25 23 33

27 25 59

39 65 73

45 94 82

94

10 14 73 25 23

13 27 94 33 39

25 59 94 65 82

45

### Algorithm

Divide A[n] to small groups, if each group has 5(step) elements, then A[0,5..,5i] is a group, sort it.

First round ie(step 5), then sort A[0,5,..,5i] A[1,6,..,5i+1] … A [4,9,..,5i+4]

Second round ie(step 3), then sort A[0,3,..,3i] A[1,4,..,3i+1] A[2,5,..,3i+2]

…

Last round step(1), then sort A[0..n-1]

### Pseudocode

Shell Sort A[n]

//Calculate step

while (step < n)

step = step\*3 + 1

//sort each divided groups using each step

while (step > 0)

for i = step to n

for (j = i; j >= n; j = j – step)

if A[j] < A[j-step]

swap(A[j], A[j-step])

step = step / 3

## Quick Sort

key =A[0]

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | i | j | action |
| init | 49 | 38 | 65 | 97 | 76 | 13 | 27 | 0 | 6 | - |
| 1 | **27** | 38 | 65 | 97 | 76 | 13 | **27** | 0 | 6 | A0=A6 |
| 2 | 27 | 38 | **65** | 97 | 76 | 13 | **65** | 2 | 6 | A6=A2 |
| 3 | 27 | 38 | **13** | 97 | 76 | **13** | 65 | 2 | 5 | A2=A5 |
| 4 | 27 | 38 | 13 | **97** | 76 | **97** | 65 | 3 | 5 | A5=A3 |
| 5 | 27 | 38 | 13 | 97 | 76 | 97 | 65 | 3 | 4 | - |
| 6 | 27 | 38 | 13 | 97 | 76 | 97 | 65 | 3 | 3 | - |
| 7 | 27 | 38 | 13 | **49** | 76 | 97 | 65 | 3 | 3 | A3=key |

### Algorithm

For array A[n]  
Set A[0] as pivot, put elements less or equal than pivot to the left, elements greater than pivot to the right.

Then recuse the left and the right.

### Pseudocode

Quick Sort A[n]

quickSort(A[n], low, high){

int partitionIndex = partition(A[n], 0, n-1);

quickSort (A[n], 0, partitionIndex-1);

quickSort (A[n], partitionIndex+1, n-1);

}

partition(A[n], low, high)

While (low < high)

For j = high to low if (A[high] <= key) A[low] = A[high] break

For i = low to high if (A[low] > key) A[high] = A[low] break

A[low] = A[high] = key

## Merge Sort

for array [6 202 100 301 38 8 1]

[6 202 100 301 38 8 1]

[6] [202] [100] [301] [38] [8] [1]

[6 202] [100 301] [8 38] [1]

[6 100 202 301] [1 8 38]

[1 6 8 38 100 202 301]

### Algorithm

Divide array into smaller groups then merge smaller groups recursive.

### Pseudocode

Merge Sort A[n]

mergeSort (A[n]){

left = A[0, n/2]

right = A[n/2, n]

sortedLeft = mergeSort(left)

sortedRight = mergeSort(right)

result = merge(sortedLeft, sortedRight)

}

merge (left, right){

//merge two sorted array

while (i < lenL && j < lenR )

result[k++] = left[i] <= right[j] ? left[i++] : right[j++];

//if left array reach the last element first

//then put the remaining elements of right array into result array

 //vice versa

}

## Summary

Random Array size 1000/10000/100000

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Avg(ms) |
| bubble | 10 | 10 | 10 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 9 |
| 232 | 232 | 222 | 222 | 234 | 222 | 212 | 222 | 230 | 222 | 225 |
| 21416 | 21936 | - | - | - | - | - | - | - | - | - |
| insertion | 10 | 10 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 7 |
| 80 | 80 | 80 | 80 | 80 | 80 | 80 | 80 | 80 | 90 | 81 |
| 7994 | 9069 | - | - | - | - | - | - | - | - | - |
| merge | 0 | 0 | 0 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 10 | 10 | 20 | 20 | 10 | 20 | 10 | 10 | 10 | 10 | 13 |
| 50 | 50 | 43 | 33 | 43 | 43 | 35 | 43 | 43 | 31 | 41 |
| quick | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | 1 |
| 21 | 10 | 14 | 20 | 21 | 9 | 30 | 10 | 20 | 10 | 16 |
| 19 | 46 | 41 | 43 | 42 | 43 | 41 | 28 | 43 | 31 | 37 |
| selection | 0 | 10 | 10 | 10 | 10 | 10 | 0 | 10 | 10 | 10 | 8 |
| 151 | 132 | 124 | 131 | 139 | 139 | 132 | 141 | 131 | 131 | 139 |
| 11933 | 12909 | - | - | - | - | - | - | - | - | - |
| shell | 0 | 10 | 10 | 0 | 0 | 1 | 0 | 0 | 10 | 10 | 4 |
| 152 | 140 | 150 | 150 | 150 | 150 | 140 | 140 | 150 | 140 | 146 |
| 14283 | 15232 | - | - | - | - | - | - | - | - | - |