Pseudocode

func bubblesort2( var a as array )

for i from 2 to N

swaps = 0//we use one flag variable swaps which will help us see //if any swap has happened or not.If no swap has occurred the array does //not need to be sorted and we come out of the loop

for j from 0 to N - 2

if a[j] > a[j + 1]//compares adjacent elements

swap( a[j], a[j + 1] )//swaps them

swaps = swaps + 1

if swaps = 0// if no swaps are made than the list is already //sorted so we get out of the loop

break

end func

b)worst case time complexity

Any element can only move left once during swapping but an element can move to the right many times. We can therefore conclude that the in the worst case, Bubble Sort does not return before performing all n iterations of the outer loop. We can figure out the worst-case runtime complexity of Bubble Sort for a list of length n by counting how many times the inner block repeats. At iteration 0, the block runs for n-1-0 times At iteration 1, the block runs for n-1-1 times At iteration n-1, the block runs for n-1-(n-1)=0 times. So in total, the block runs ∑n−1i=0(n−i−1) times

∑n−1i=0(n−i−1)=n2−∑n−1i=0i−n=n2−n(n−1)/2−n=n2−n2/2+n/2−n=n2/2−n/2

(To compute ∑n−1i=0i , we use the fact that ∑mj=1j=m(m+1)/2 , and substitute m=n−1)

We can therefore conclude that the worst-case runtime complexity of Bubble Sort if O(n^2) , just like Selection Sort.

Best case time complexity

void bubble\_sort(int a[],int n)

{

int i,j;

int flag=1;

for(i=0;i<n-1;i++)

{

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

{

if(a[j]>a[j+1])

{ flag=0;

swap(a[j],a[j+1]);

}

}

if(flag==1)

{

cout<<"already sorted"<<endl;

break;

}

}

for(i=0;i<n;i++)

cout<<a[i]<<" ";

cout<<endl;

}

Now if the array is already sorted, then flag=1 at the end of the first pass. So when this condition is met, we break out of the loop. Since we only pass through n-1 elements in the best case, the complexity is linear as opposed to the Quadratic Complexity of Bubble-Sort.

So it has O(n) complexity in Best Case.

Average case time complexity

In average case the given array is partially sorted

Average case complexity = worst case+best case/2

=(cn+cn^2)/2

If n is very large than cn^2 would be the dominating term so time complexities for

Average case would me O(n^2)

c) Heap sort is not a stable sorting algorithm as the order of the original elements is changed when heapify method is used. The order in which they are printed depends upon on the node that they are assigned to when heapify method is used and may change.

However Insertion, Merge and bubble sort are stable since they use a comparison like:

if A[i] > A[i+1] then………….finish

the keys are the same and the one that comes before stays in that position.

d)In insertion, bubble and heap sort a comparison is made before doing anything to check if there is actually a need to do something and the array is not already sorted hence these algorithms are adaptive but in the merge sort the array is split before comparing the elements and hence it cannot be considered as an adaptive sorting algorithm.