Sorting in Arrays

Sorting an array means to arrange the elements in the array in a certain order. Various algorithms have been designed that sort the array using different methods. Some of these sorts are more useful than the others in certain situations.

Terminologies

Internal/External Sorting

Internal sorting means that all the data that is to be sorted is stored in memory while sorting is in progress.

External sorting means that the data is stored outside memory (like on disk) and only loaded into memory in small chunks. External sorting is usually applied in cases when data can’t fit into memory entirely, effectively allowing to sort data that does not fit in the memory.

Stability of Sort

A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in the sorted output as they appear in the unsorted input.

A sorting algorithm is said to be unstable if there are two or more objects with equal keys which don’t appear in same order before and after sorting.

Bubble Sort

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order. The pass through the list is repeated until the list is sorted.

6 5 3 1 8 7 2 4

Bubble Sort Animation

Swfung8 [CC-BY-SA 3.0]

This is an inefficient sort as it has to loop through all the elements multiple times. It takes O(n^2) time to completely sort the array.

Code for Bubble Sort

```c
#include <stdio.h>

int main()
{
  int arr[100], n, i, j, temp;

  printf("Enter number of elements\n");
  scanf("%d", &n);

  printf("Enter %d integers\n", n);

  for (i = 0; i < n; i++)
    scanf("%d", &arr[i]);

  for (i = 0 ; i < n - 1; i++)
  {
    for (j = 0 ; j < n - i - 1; j++)
    {
      if (arr[j] > arr[j+1])
      {
        temp = arr[j];
        arr[j] = arr[j+1];
        arr[j+1] = temp;
      }
    }
  }

  printf("Sorted list in ascending order:\n");

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

```
31.       printf("%d\n", arr[i]);
32.
33.   return 0;
34. }</stdio.h>
```

Properties

- **Average Time Complexity**: O(n^2)
- **Stability**: Stable

Best use case

This is a very elementary sort which is easy to understand and implement. It is not recommended in actual production environments. No external memory is required to sort as it is an in-place sort.

Insertion Sort

In insertion sort, every iteration moves an element from unsorted portion to sorted portion until all the elements are sorted in the list. An analogy of insertion sort is the sorting of a deck of cards with our hands. We select one card from the unsorted deck and put it in the right order in our hands, effectively sorting the whole deck.

Steps

1. Assume that first element in the list is in its sorted portion of the list and remaining all elements are in unsorted portion.
2. Take the first element from the unsorted list and insert that element into the sorted list in order specified (ascending or descending).
3. Repeat the above process until all the elements from the unsorted list are moved into the sorted list.

6 5 3 1 8 7 2 4

Insertion Sort

Code for Insertion Sort

```
#include<stdio.h>

int main()
{
   int data[100],n,temp,i,j;
   printf("Enter number of elements to be sorted:");
   scanf("%d",&n);
   printf("Enter elements: ");
   for(i = 0; i < n; i++)
      scanf("%d",&data[i]);
    for(i = 1; i < n; i++)
    {
      temp = data[i];
      j = i - 1;
      while(temp < data[j] && j>=0)
      {
        data[j + 1] = data[j];
        j = j - 1;
      }
      data[j + 1]=temp;
    }
    printf("Sorted array: ");
    for(i = 0; i < n; i++)
      printf("%d  ",data[i]);
      return 0;
}
</stdio.h>
```

Properties

- **Average Time Complexity**: O(n^2)
- **Stability**: Stable

Best use case

Although this is a elementary sort with the worst case of O(n^2), it performs much better when the array is nearly sorted, as lesser elements would have to be moved. It is also preferred when the number of elements are less as it has significantly less overhead than the other sorts. It consumes less memory and is simpler to implement.

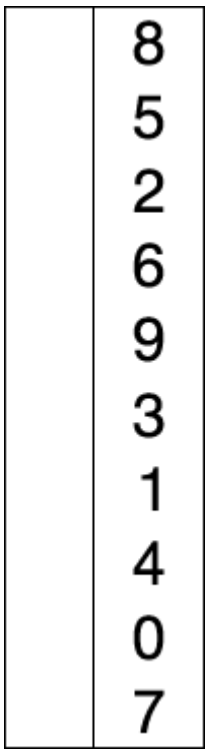
In some quick sort implementations, insertion sort is internally to sort the smaller lists faster.

Selection Sort

Selection sort is generally used for sorting files with very large records and small keys. It selects the smallest (or largest) element in the array and then removes it to place in a new list. Doing this multiple times would yield the sorted array.

Steps

1. Select the first element of the list.
2. Compare the selected element with all other elements in the list.
3. For every comparison, if any element is smaller (or larger) than selected element, swap these two elements.
4. Repeat the same procedure with next position in the list till the entire list is sorted.



Selection Sort

Code for Selection Sort

```
1. //
2. // DS Handbook
3. // Selection Sort
4. //
5.
6. #include <stdio.h>
7.
8. int main()
9. {
10.   int array[100], n, pos, temp, i, j;
11.
12.   printf("Enter number of elements\n");
13.   scanf("%d", &n);
14.
15.   printf("Enter the %d values\n", n);
16.
17.   for (i = 0; i < n; i++)
18.     scanf("%d", &array[i]);
19.
20.   for (i = 0; i < (n - 1); i++)
21.   {
22.     pos = i;
23.
24.     for (j = i + 1; j < n; j++)
```

```
    {
      if (array[pos] > array[j])
        pos = j;
    }
    if (pos != i)
    {
      temp = array[i];
      array[i] = array[pos];
      array[pos] = temp;
    }
  }

  printf("Sorted list in ascending order:\n");

  for (i = 0; i < n; i++)
    printf("%d\n", array[i]);

  return 0;
}</stdio.h>
```

Properties

- **Average Time Complexity:** O(n^2)
- **Stability:** Non Stable

Best use case

This sort is not influenced by the initial ordering of elements in the array and can be sued to efficiently sort small lists. It performs the least amount of data movement amongst all sorts, therefore it could be used where data manipulation is costly.

Quick Sort

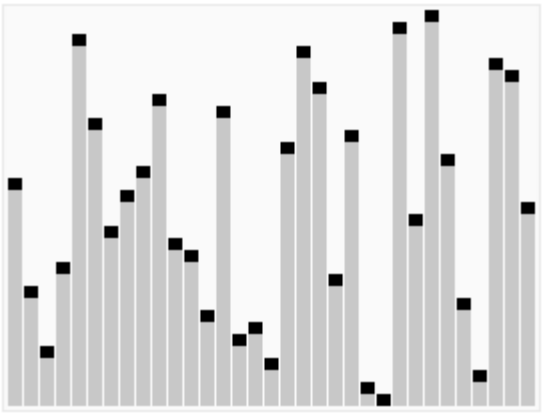
Quick Sort is an efficient divide-and-conquer algorithm. It divides a large list into two smaller sub-lists based on a pivot chosen, into smaller and larger elements. Quick Sort then recursively does this to the sub-lists finally producing a sorted list.

Steps

1. Pick an element, called the pivot.
2. 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. After this partitioning, the pivot is in its final position. This is called the partition operation.

3. 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.



Quick Sort Visualisation

RolandH [CC-BY-SA-3.0]

Code for Quick Sort

```c
#include <stdio.h>

void swap(int a, int b)
{
  int t = a;
  a = b;
  b = t;
}

int partition (int arr[], int low, int high)
{
  int pivot = arr[high];    // pivot
  int i = (low - 1);  // index of smaller element

  for (int j = low; j <= high- 1; j++)
  {
    // if current element is smaller than the pivot
    if (arr[j] < pivot)
    {
      i++;    // increment index of smaller element
      swap(&arr[i], &arr[j]);
    }
  }
  swap(&arr[i + 1], &arr[high]);
  return (i + 1);
```

```
}

void quick_sort(int arr[], int low, int high)
{
  if (low < high)
  {
    int pi = partition(arr, low, high);
    quick_sort(arr, low, pi - 1);
    quick_sort(arr, pi + 1, high);
  }
}

int main()
{
  int a[100], n, i;
  printf("No. of elements to sort");
  scanf("%d", &n);
  printf("\nEnter the elements:\n");

  for(i = 0; i < n; i++)
    scanf("%d", &a[i]);

  quick_sort(a, 0, n - 1);
  printf("\nArray after sorting:");

  for(i = 0; i < n; i++)
    printf("%d ",a[i]);

  return 0;
}</stdio.h>
```

Properties

- **Average Time Complexity:** O(n log n)
- **Stability:**Non Stable (Stable versions are available)

Best use case

This is one of the best performing sorts that can sort any data most efficiently. It uses in-place sorting which means it has the best cache locality and does not use any extra memory to perform the sort. If the pivot choosing method is effective, this sort is one of the most versatile sorts out of all.

Merge Sort

Merge sort is a very efficient comparison-based sorting algorithm. It is a divide-and-conquer algorithm, which works by repeatedly dividing the array in small parts and merging them again in the sorted order.

Steps

1. Divide the unsorted list into *n* sub-lists, each containing 1 element.
2. Repeatedly merge the sub-lists to produce new sorted sub-lists until only 1 sub-list remains. This will be the final sorted list.Code for merge sort

6 5 3 1 8 7 2 4

Merge Sort Animation

Code for Quick Sort

```
1. #include<stdio.h>
2.
3. void merge(int a[], int i1, int j1, int i2, int j2)
4. {
5.   int temp[50];     // temporary array used
6.   int i, j, k;
7.   i = i1;           // beginning of the first list
8.   j = i2;           // beginning of the second list
9.   k = 0;
10.
11.   while (i <= j1 && j <= j2)     // while elements in
  both lists
12.   {
13.     if(a[i]<a[j]) temp[k++]="a[i++];" else="" }="" 
  while(i="" <="
```