Chapter 10

Sort and Search

Sort

$$\{2, 3, 1, 7, 16, 14, 8, 20, 12, 7\}$$

in increasing order.

The algorithms
Bubble sort
Selection sort
Merge sort
Quick sort

There is incremental sort
Bubble
Selection

And divide and conquer
Merge
Quicksort

10.1 Sorting and Searching algorithms

Basic computational task - sorting data.
Eg

Numbers in ascending/descending order

names alphabetically

A wide variety of sorting algorithms rely on 2 basic operation

- Compare 2 items to see which is largest
- Copy (swap) 1 item from one place to another

Sorting ⇒ perform (1) and (2) repeatedly

Some ways of combining (1) and (2) are faster/ more efficiently than others..

10.1.1 Bubble sort code

```
#include <stdio.h>
#define N 8

void bubble(int a[], int n);
void swap( int *x, int *y);

main()
{
  int a[N]={7,3,66,3,-5,22,-77,2}, i;

  bubble(a,N);
```

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

}

void bubble(int a[], int n)
{
  int i,j,k;
  for ( i=0;i< n-1; i++){
   for (j=n-1;j>i; j--)
    if ( a[j-1] > a[j] )
     swap( &a[j-1], &a[j]);

  for (k=0;k<N;k++)
   printf("%d\t", a[k]);

   printf("\n");
}
}

void swap( int *x, int *y)
{
  int tmp;

  tmp = *x;
  *x = *y;
  *y = tmp;
}
```

Unordered

7 3 66 3 -5 22 -77 2

at start of 1st pass is i=0: a[6] is compared to a[7]. these are in order so no swap. Then a[5] compared with a[6], not in order so swapped. Continue

until a[0], a[1] are compared. We get

1st pass -77 7 3 66 3 -5 22 2

for the 2nd pass: a[0] is unchanged, ie i=1 and we compare from a[7] to a[1].

2nd pass -77 -5 7 3 66 3 2 22

3rd pass -77 -5 2 7 3 66 3 22

4th pass -77 -5 2 3 7 3 66 22

5th pass -77 -5 2 3 3 7 22 66

6th pass -77 -5 2 3 3 7 22 66

7th pass -77 -5 2 3 3 7 22 66

Note the array is sorted after the fifth pass. It is possible to modify the algorithm to terminate if no swaps were made in a given pass.

In general bubble sort is very inefficient. It is an N^2 algorithm.

This means that the running time of the order N^2, where N is the number of things to sort.

10.1.2 Selection Sort

Find the largest item - move it to the end of the list. Repeat for remaining items.

To find the largest item, the computer moves through the list one item at a time keeping track of the largest number.

Eg.

7 3 66 3 -5 22 -77 2

1st pass 7 3 *2* 3 -5 22 -77 *66*

2nd pass 7 3 2 3 -5 *-77* *22* 66

```
/* this program merges 2 arrays, the first has 2 elements and the second
   array has 3. I have initialised the arrays with values so that you can run
   the code and see the output. This can of course be generalised to arrays
   of any size which are read from a file or from the keyboard.
*/
#include <stdio.h>
#define M 2
#define N 3
main()
{

 int i=0,j=0,k=0;
 int a[M] = {2,3}, b[N]={4,5,6}, c[M+N];

 while (i<M && j<N)
 if( a[i] < b[j] )
   c[k++] = a[i++];
 else
   c[k++] = b[j++];

 while( i<M )
  c[k++] = a[i++];
 while ( j<N )
  c[k++]=b[j++];

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

}
```


10.2 Merge and Merge-sort

10.2.1 Merge

To merge two ordered arrays a[], b[] into one ordered array c[].
Example

1. compare a[0] and b[0]
   whichever is smaller, eg b[0] put in c[0]
2. compare a[0] and b[1] whichever is smaller put in x[1], eg b[1] put in c[1]
3. compare a[0] and b[2] put smallest into c[2] eg a[0] in c[2]
4. and so on, and so on

eventually you get to the end of a[] or b[], eg b[]
Finally copy remaining elements of a[] into c[]
NB. It is the programmers responsibility to make sure c[] is large enough.

10.2.2 Merge-sort

We use the 'merge' function above to sort the array.
Consider the array

4	3	1	67	55	8	0	4	-5	37	7	4	2	9	1	-1
3	4	1	67	8	55	0	4	-5	37	4	7	2	9	-1	1
1	3	4	67	0	4	8	55	-5	4	7	37	-1	1	2	9
0	1	3	4	4	8	55	67	-5	-1	1	2	4	7	9	37
-5	-1	0	1	1	2	3	4	4	4	7	8	9	37	55	67

Clearly Merge-sort is more efficient than bubble sort - less passes through array required.
Merge-sort is an $NlogN$ algorithm (where N is the size of the array)
Recall Bubble sort is N^2 algorithm
For large N

$$NlogN << N^2$$

10.3 Quicksort

On most machines for large N it is the fastest known algorithm
-it is also the most complicated
Main idea

- Remove one item from the list - call it 'a'
- Divide the remaining items into 2 parts
  - items bigger than a
  - items smaller than a
- Move all smaller items to the beginning of the list and all bigger items to the end of the list, leaving a gap in the middle
- Put 'a' in the gap. It is now in its correct place and shouldn't be moved again
- a now divides the list into 2 parts - each must be sorted.
- Apply Quicksort algorithm recursively to each sub-list until sorted

The cleverness of Quicksort is the way it efficiently divides the list into smaller and bigger items.

Algorithm

It often uses a random number generator to pick the array element a. uses pointers to scan left and right of a and a swap function to get things to the right side of a.
All the details are in Numerical recipes in C or you could just cheat and use predefined `qsort` function the C standard library.
The function looks like

```
void qsort(void *array, int n_elts,
           int elt_size, int compare(*p,*q) );
```

ie 4 inputs

1. the array to be sorted
2. number of elements in the array
3. the size of in bytes of each element
4. the name of the function that compares array elements

Example

```
#include <stdlib.h>
#include <stdio.h>
int cmp(*vp,*vq);
main()
{
double a[6] = {1,-2,4,7,-3,-2};
qsort(a, 6, sizeof(double), cmp );
}
```