# Sorting and Searching

Some examples are included for reference, but may more exist.

```
//-------------------------------------------------------------------------

   void swapslots(emprecord  employee[], int size, int i, int j )
   { emprecord temp;
       temp=employee[i];
     employee[i]=employee[j];
     employee[j]=temp;
   }

//-------------------------------------------------------------------------

   void bubblesortarrayversion1(emprecord  employee[], int size)
   {  int i;
      bool swappedrecord;

      do
        {
          swappedrecord =false;
          for (i=0;i<(size-1);i++)
           {
            if (employee[i].name > employee[i+1].name)
              { swapslots(employee, size, i, i+1);
                swappedrecord=true;
              };
           }
        }
     while (swappedrecord !=false);

   }

//-------------------------------------------------------------------------

   void bubblesortarrayversion2(emprecord  employee[], int size)
   {  int i;
      bool swapped;

    for (int i = 0; i < size; ++i)
    {
        bool swapped = false;
        for (int j = 0; j < size - (i+1); ++j)
```

- This function exchanges the objects in slots I and j of the array. This function does not know which sorting algorithm has called it or which fields have been compared to swap.

- This bubble sort function is basically the same used in the first class.

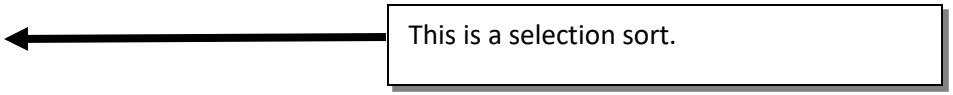
- This is another version of a bubble sort.

```
        {
            if (employee[i].name > employee[i+1].name)
            {
                swapslots(employee, size, j, j+1);
                swapped = true;
            }
        }

        if (!swapped) break;
    }
}

//--------------------------------------------------------------------------

void SelectionSort(emprecord  employee[], int size)
{  string name;
   int i, startscan, min;
   string minvalue;
    for ( startscan = 0; startscan < (size-1); startscan++)
    {
        min = startscan;
        minvalue=employee[startscan].name;
        for (int j = startscan+1; j < (size); j++)
        {
            if (employee[j].name < name)
            {
                min = j;
                minvalue=employee[j].name;
            }
        }
        swapslots(employee, size, startscan, min );
    }
}
```



```
//---------------------------------------------------------------------------

//useful for small and mostly sorted lists
//expensive to move array elements

void InsertionSort(emprecord  employee[], int size)

{ emprecord save;
  for (int i = 1; i < size; ++i)
    {
        bool inplace = true;
        int j = 0;
        for (; j < i; ++j)
        {
            if (employee[i].name < employee[i+1].name)
            {
                inplace = false;
                break;
            }
        }

        if (!inplace)
        {
            save = employee[i];
            for (int k = i; k > j; --k)
            {
                employee[k] = employee[k-1];
            }
            employee[j] = save;
        }
    }
}
```

This is an insertion sort.


```
//useful for small and mostly sorted lists
//expensive to move array elements
```


It finds the slot to insert and moves the rest of the array to the next slot.

```
//-------------------------------------------------------------------------

   void binarySearch(emprecord employee[], int size)
   { string name;
     cout << "\n\n Enter Name to search for ";
     getline(cin,name);

       int i, first=0,
           last=size-1,
             middle,
             position=-1;
       bool found =false, exit=false;
       while ( !found && first <= last)
         { middle=(first+last)/2;                  // Calculate midpoint of the remaining section of the array to
be searched
          cout << "First  : "<< first<< "  Last  : "<< last<<" Middle  : "<< middle<<endl;
           if (employee[middle].name==name)
             {  found=true;
              position=middle;
               }
               else if (employee[middle].name>name)   // The value is he lower half of the Calculate midpoint of
the remaining section of the array to be searched
                       last=middle-1;
               else first=middle+1;
         }
     if (!found)
        cout <<"\n\nName not found\n\n";
        else cout <<"\n\nName found at slot "<<position<<endl;
     cin.get();
   };
```

This is an insertion sort.

```
//useful for small and mostly
sorted lists
//expensive to move array
elements

It finds the slot to insert and
moves the rest of the array to
the next slot.
```