Group Name: BOBOBOY

Group Member:

1. AERON GOH MING LUN A22EC0033
2. TAY SHUN WEI A22EC0110
3. LIN CHONG HUI A22EC0184

Course: DATA STRUCTURE AND ALGORITHMS
Section: 04

Lecturer : Ms. LIZAWATI BINTI MI YUSUF

Table of contect

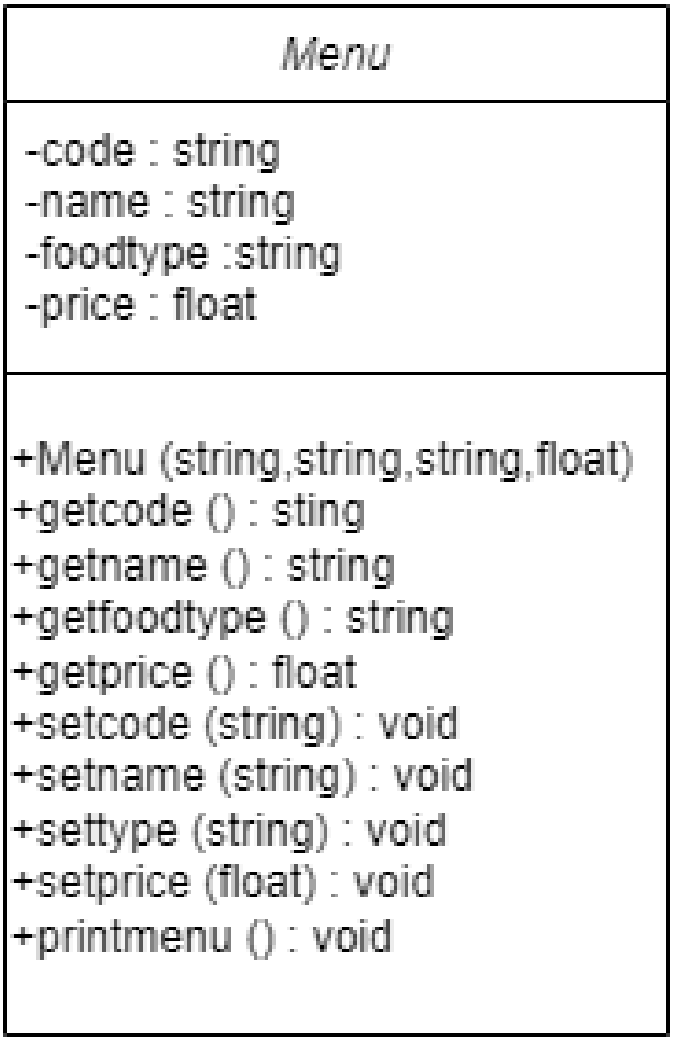
Objective

The main objective of BOBOBOY's Restaurant Management System (RMS) is to optimize the order management system with effectiveness features and to minimize the time to get through our ordering system. Our system is created for a full process from ordering to payment session and it can be done via through the system and the customers can get the food in the best condition without wasting any time on waiting or getting used to the system.
All the features, panel and functions are to make sure the customers have good feedback to our restaurant and satisfy not only about the food, but also the whole system and environment of the BOBOBOY restaurant.

Synopsis

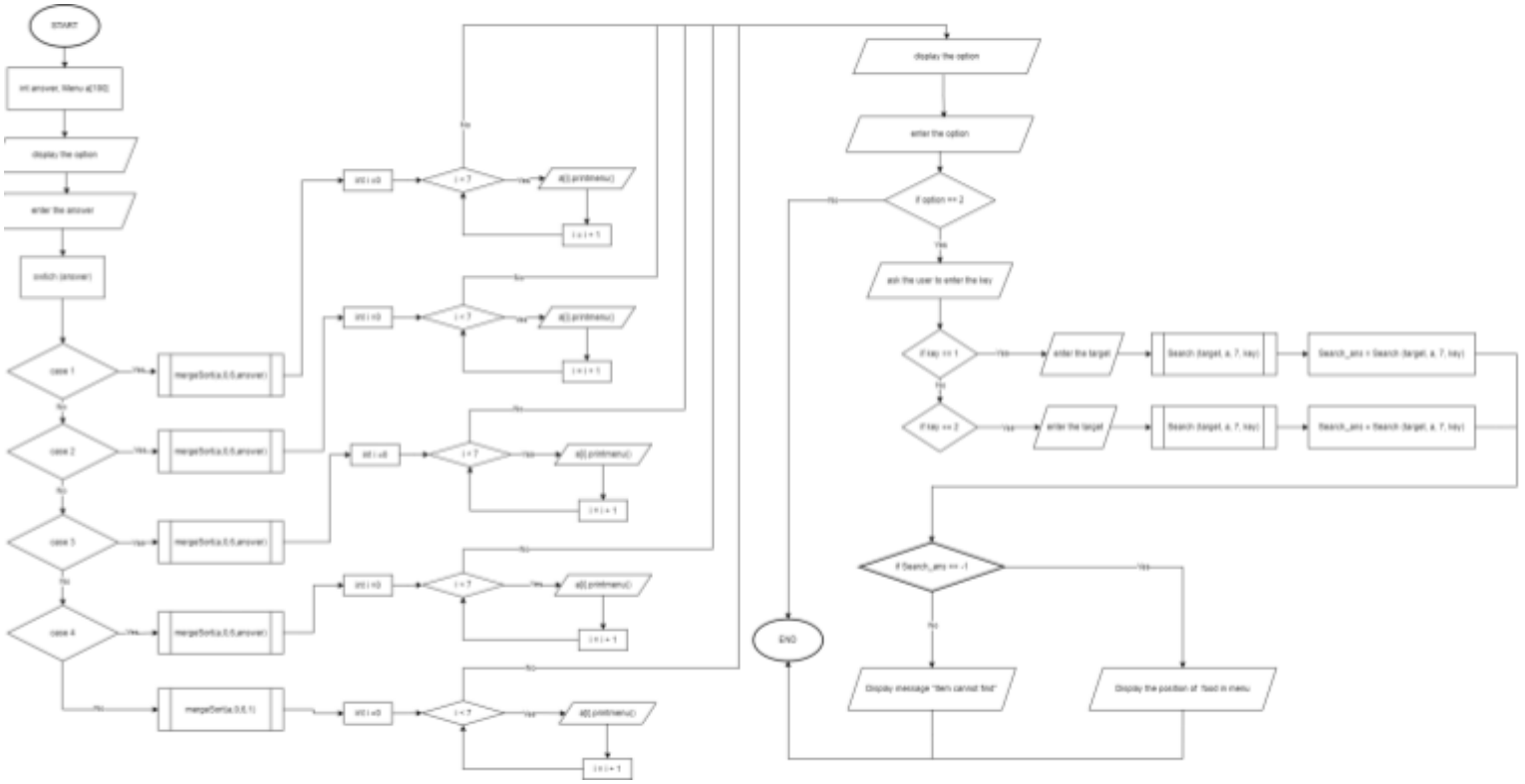
In a restaurant, it is most important that RMS should be user friendly and show a detailed menu in a simple way. Besides, the customer's emotion is very important as they want to get the food they ordered as fast as possible. In BOBOBOY restaurant, the RMS contained two main functions in the menu which is sorting and searching. The purpose of implementing these two functions is because we want the customers to look through the menu based on their favorite arrangement such as food name with ascending alphabet arrangement. . After that, if they cannot find what specific food they want on the menu, they can type the details or info about that food on the search panel and it will show the position of that product so next time they can straight away order from the menu without wasting time.. All of these are to make our customers like the ordering system of BOBOBOY restaurants and they feel easy and simple when ordering their desired food.

Class diagram

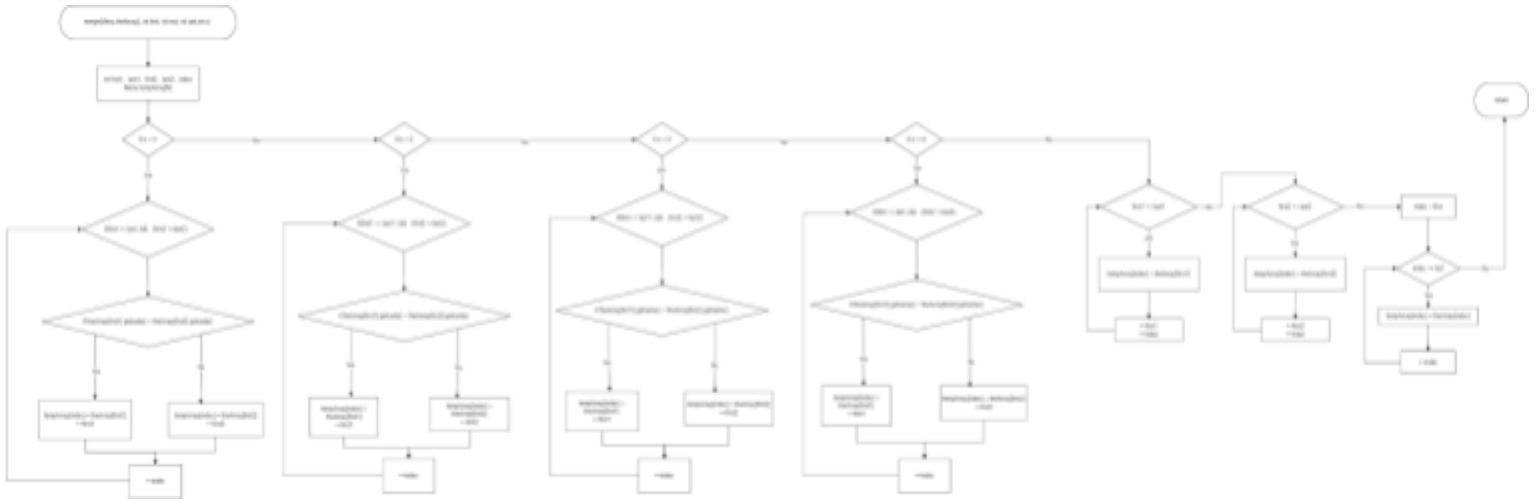


Flowchart

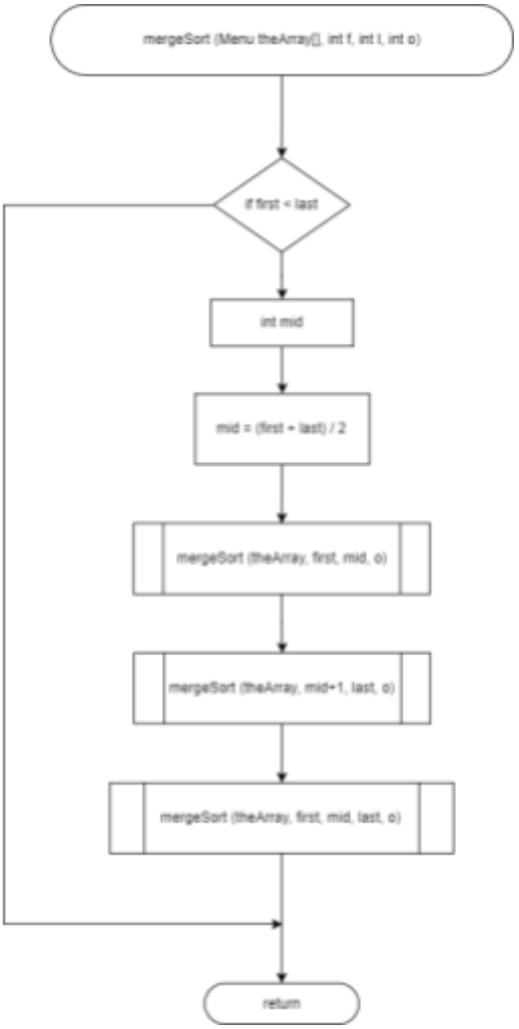
Main function



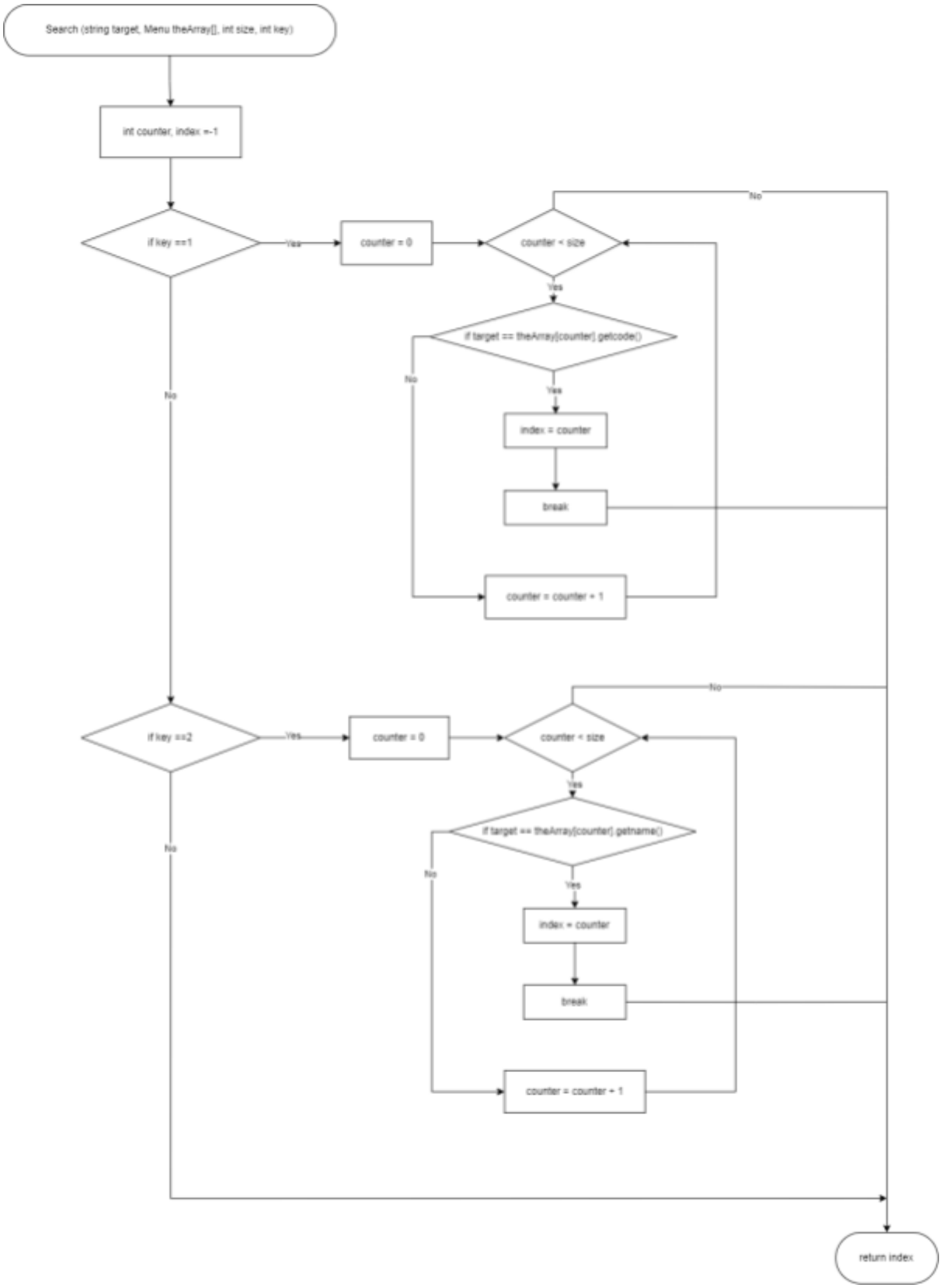
Merge function



MergeSort function



Search function



Description of Sorting and Searching Process:

Based on the provided class diagram, the class named "Menu" contains four attributes which are code**, name, type,** and **price**. We choose merge sort for our sorting method in class Menu. In our RMS, the system allows users to sort the menu based on their desired mode such as ascending based on food id or descending and ascending or descending based on the food name. If the user does not want to choose any type, the system will show the menu in default order.

In the menu, if users cannot find the food they want on that page in the system, the searching function is provided. The system can detect food name and id in our searching function. If their search result is available, the system will display the position of the food in the specific pages in the menu. If not, it will be blank and users need to search again.

Coding

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <string>
#define N 7
using namespace std;

class Menu{
	private:
		string code,name, foodtype;
		float price;
	public:
		Menu(string c = " ", string n = " ", string f = " ", float p =0.0 ){
			code = c;
			name = n;
			foodtype = f;
			price = p;
		}
		string getcode(){
			return code;
		}
		string getname(){
			return name;
		}
		string getfoodtype(){
			return foodtype;
		}
		float getprice(){
			return price;
		}
		void setcode(string c){
			code = c;
		}
		void setname(string n){
			name = n;
		}
		void settype(string t){
			foodtype = t;
		}
		void setprice(float p){
			price = p;
		}
```

```
                void printmenu(){
                cout << code << setw(20) << name << setw(20) << foodtype << setw(20) <<
price << endl;
                }
};

void merge(Menu theArray[], int first, int mid, int last,int o) {
   Menu tempArray[7];
   int first1 = first;
   int last1 = mid;
   int first2 = mid + 1;
   int last2 = last;
   int index = first1;
   if(o == 1){
   for (; (first1 <= last1) && (first2 <= last2); ++index) {
      if (theArray[first1].getcode() < theArray[first2].getcode()) {
         tempArray[index] = theArray[first1];
         ++first1;
      } else {
         tempArray[index] = theArray[first2];
         ++first2;
      }
   }
         }
         else if(o == 2){
         for (; (first1 <= last1) && (first2 <= last2); ++index) {
      if (theArray[first1].getcode() > theArray[first2].getcode()) {
         tempArray[index] = theArray[first1];
         ++first1;
      } else {
         tempArray[index] = theArray[first2];
         ++first2;
      }
   }
         }
         else if(o == 3){
         for (; (first1 <= last1) && (first2 <= last2); ++index) {
      if (theArray[first1].getname() < theArray[first2].getname()) {
         tempArray[index] = theArray[first1];
         ++first1;
      } else {
         tempArray[index] = theArray[first2];
         ++first2;
      }
```

```
    }
        }

        else if(o == 4){
        for (; (first1 <= last1) && (first2 <= last2); ++index) {
      if (theArray[first1].getname() > theArray[first2].getname()) {
        tempArray[index] = theArray[first1];
        ++first1;
      } else {
        tempArray[index] = theArray[first2];
        ++first2;
      }
    }
        }
    for (; first1 <= last1; ++first1, ++index)
      tempArray[index] = theArray[first1];

    for (; first2 <= last2; ++first2, ++index)
      tempArray[index] = theArray[first2];

    for (index = first; index <= last; ++index)
      theArray[index] = tempArray[index];
    }

void mergeSort(Menu theArray[], int first, int last,int o) {
    if (first < last){
      int mid = (first + last) / 2;
      mergeSort(theArray, first, mid,o);
      mergeSort(theArray, mid+1, last,o);
      merge(theArray, first, mid, last, o);
    }
}
```

```
int Search(string target,Menu theArray[],int size,int key){
	int counter;
	int index = -1;
	if(key == 1 ){
		for(counter = 0;  counter < size; counter++){
			if(target == theArray[counter].getcode()){
				index = counter;
				break;
			}
		}
	}
	else if(key == 2){
		for(counter = 0;  counter < size; counter++){
			if(target == theArray[counter].getname()){
				index = counter;
				break;
			}
		}
	}
	return index;
}

Main function
int main(){
   string menu_code, menu_name, menu_type;
	float menu_price;
	Menu a [N];
	int i = 0, option;
	ifstream file("input.txt.txt");
	if(!file){
		cout << " Error opening file" << endl;
	}
	/*
	else{
		cout << "File can run" << endl; // use for testing the file
	}
	*/

	while(getline(file,menu_code,',')){
		getline(file,menu_name,',');
		getline(file,menu_type,',');
		file >> menu_price;
		file.ignore();
```

```
            a[i].setcode(menu_code);
            a[i].setname(menu_name);
            a[i].settype(menu_type);
            a[i].setprice(menu_price);
            i++;
        }
        cout << "********** Welcome to BOBOBOY Restaurant's Ordering System
**********" << endl << endl;

        int answer;
        cout << "Select menu display type: " << endl;
        cout << "1. Arrange the Menu by following the code (ascending order)" << endl;
        cout << "2. Arrange the Menu by following the code (descending order)" << endl;
        cout << "3. Arrange the Menu by following the name (ascending order)" << endl;
        cout << "4. Arrange the Menu by following the name (descending order)" << endl;
        cout << "Your choice: ";
        cin >> answer;
        cout << "-----------------------------------------------------------" << endl;
        cout << "Code" << setw(17) << "Name" << setw(20) << "Type" << setw(20) <<
"Price(RM)" << endl;
        cout << "-----------------------------------------------------------" << endl;
   switch(answer){
            case 1: mergeSort(a,0,N-1,answer);
                for(int i=0; i <N; i++){
                   a[i].printmenu();
                        }
                        break;
            case 2: mergeSort(a,0,N-1,answer);
                for(int i=0; i <N; i++){
                   a[i].printmenu();
                        }
                        break;
            case 3: mergeSort(a,0,N-1,answer);
                for(int i=0; i <N; i++){
                   a[i].printmenu();
                }
                break;
            case 4: mergeSort(a,0,N-1,answer);
                for(int i=0; i <N; i++){
                   a[i].printmenu();
                }
                break;
            default:mergeSort(a,0,N-1,1);
                for(int i=0; i <N; i++){
```

```
                        a[i].printmenu();
                                }
                                break;
        }
    cout << "Process:" << endl;
        cout << "1. Order" << endl;
        cout << "2. Search item" << endl;
        cout << "Your choice: ";
        cin >> option;
        int Search_ans;
        if(option == 2){
                int key;
                string target;
           cout << "What you want to search:" << endl;
           cout << "1. Code" << endl;
           cout << "2. Name" << endl;
           cout << "Your choice: ";
           cin >> key;
           cout << "Enter the code you want to search: ";
                cin >> target;
                Search_ans = Search(target,a,N,key);
                if(Search_ans == -1){
                        cout << "Item cannot find" << endl;
                }
                else{
                        cout << "The item is at position " << Search_ans+1 << endl;
                }
        }
}
```