**Sailing Club Project Write-up**

End user: Administrator at Bosham Sailing Club

Every year at Bosham Sailing Club (BSC) there is a junior sailing regatta that approximately 150 people take part in. The regatta takes place over the course of a week and the parents of the competitors must do two duties over the course of the week. At night there are social that the competitors can attend. This program will aid the administrator of the club to enter and store the details of all the competitors that are signing up for the regatta.

The program will store the competitor details such as their name, age and what type of boat they are sailing as well as who they may be sailing with. It also stores data about the duties the parents have signed up to and the number of tickets they need for the socials as different people may want to miss certain socials or get tickets for relatives or friends.

The data will then be output in the form of a table and the user will also have the option to view and edit the data in more detail in another window. The admin can then sort the data so that other people in the club can easily view it.

Analysis:

Users:

There will be one main user who is the club admin. This admin will be able to add new data and edit existing data and they also will have the power to add new admins, these new admins may be the commodore or rear commodore of cadets. Other users, such as the heads of the classes and the safety co-ordinator, can look at the data but cannot edit it so they can plan aspects of the regatta that they are in charge of.

Current system:

The current system is paper based. Competitors fill out a paper form and hand it into the office, which the admin then files away. The admin then the key data into an excel spreadsheet however if they want to find the full details about a specific person then they have to look through all the sheets of paper to find them. Theoretically only people who work in the office can get to this data however the office is often unlocked allowing random people to get in and edit the data.

Input:

Currently the competitors fill in a form which is then given to the administrator who then files away the sheets in a drawer in the club office.

Processing:

The data needs to be searched and sorted and the ages of the competitors, all the data needs to save onto the computer and the input data needs to be checked for errors.

Outputs:

My system will digitalise the current system allowing the key details to be seen in a table form and the rest of the data will be able to be searched through. It will also be password protected so only the admin or other users that the admin gives access to will be able to edit the data. Anyone can look at the data so people who need to access the data but who do not need to edit it can do so. The data can also be backed up to an external data store.

Volumetric:

There are approximately 200 competitors taking part and there are 18 fields that data is stored in. The total data stored should be approximately 60KB. This data will be stored in RAM while the program is running however this should not have any impact on the performance of the computer. This data being stored is also very reasonable and will not have a large impact on the overall capacity of the computer’s hard-drive.

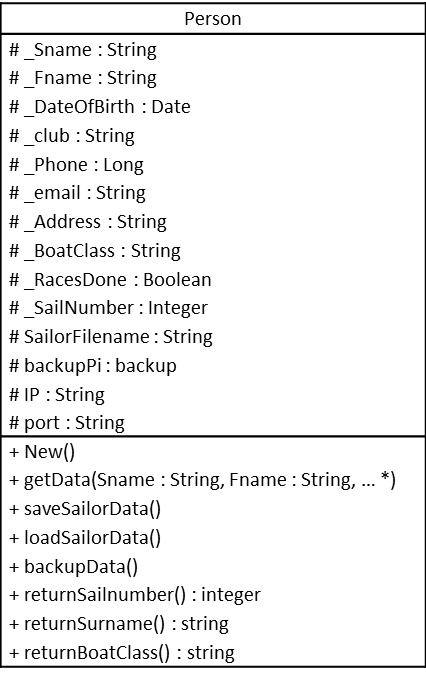
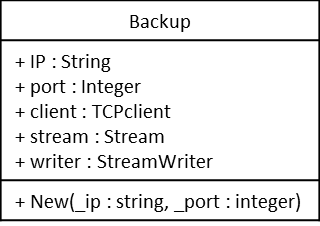
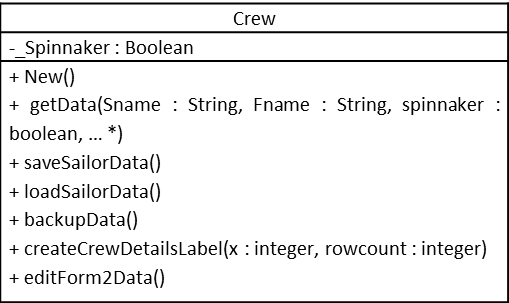
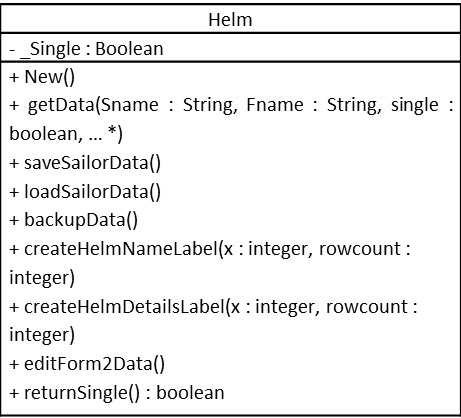
Problems with current system:

The problem with the current system is that it is very hard for the organisers of the regatta to view the data that they need to. They need to sort through hundreds of competitors to find the data for just one of them. My system will allow them to search quickly and easily for competitors and sort the data to make it easier to look at lots of data quickly.

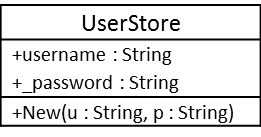
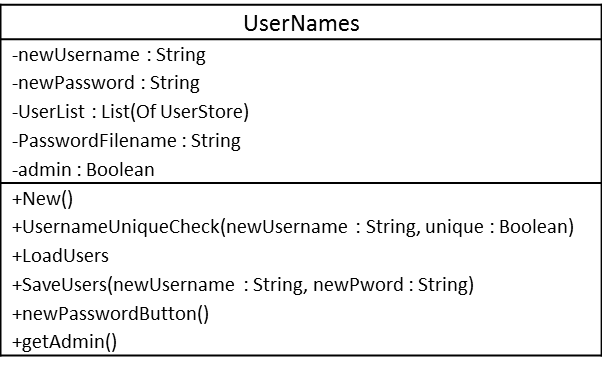
Another problem is that the data is not very secure so can be edited easily; my system will add another layer of security to the data to prevent tampering with the data.

Requirements:

|  |  |  |
| --- | --- | --- |
| **ReID** | **Objective** | **Description** |
| R01 | Input Data | Input the data for the helm and the crew |
| R02 | Store Helm Data | Store the helm data in a list |
| R03 | Store Crew Data | Store the crew data in a list with the sail number linking it to the helm |
| R04 | Validate Data | Validate all the data and make the user re-enter any faulty data |
| R05 | Output Data to a table | Output the key data to a table for easy access |
| R06 | Output all data | Output the rest of the data in a separate form |
| R07 | Sorts and searches | Allow users to search for individual users or to sort the data making data gathering easy |
| R08 | Save data | Permanently save the data |
| R09 | Load data | Load the data from memory |
| R10 | Have a login system | Add a log in so only the admin can edit the data |



\*These is the rest of the arguments that need to entered into get data



Design

Overall System Design

* Input: The administrator at Bosham Sailing Club from the data given by the competitors.
* Process: Validate all the inputted data then have the option to sort and search the data.
* Storage: store the data in a .txt file in a table for each class that is being stored.
* Output: Output the data in a table formed of the key data and a form that shows all of the data in detail

Hierarchy chart



Validation checks

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field Name | Validation check | Description | Error Message | Data | Caught |
| First Name | Type | Only allow letters | The first name contains numbers | Char1e | yes |
| Surname | Type | Only allow letters | The surname contains numbers | D0le | yes |
| Date of Birth | DoB < current date | The Date of birth must be in the past | That is an invalid date | 23/02/19 | yes |
| Phone Number | Length = 11 | The phone number must be exactly 11 characters long | The phone number is not the correct length | 07747 | yes |
| Phone Number | Type | Only allow numbers | The phone number contains letters | 077e7 | yes |
| Sail Number | Type | Only allow numbers | The sail number contains letters | A412 | yes |

Storage media and format

The data will be stored in a text file on the computer and can be backed up

The memory allocated to each data type in visual basic is allocated as such:

|  |  |
| --- | --- |
| Integer | 4 bytes |
| Long | 8 bytes |
| String | 32 bytes |
| Date | 8 bytes |
| Boolean | 1 byte |

There are 3 classes associated with each competitor: Person, helm/crew, and backup. Each competitor therefore has 20 attributes associated with them. For a single competitor the storage requirement will be:

4 bytes per integer x 2 attribute = 8 bytes

8 bytes per long x 1 attribute = 8 bytes

32 bytes per string x 10 attributes = 320 bytes

8 bytes per date x 1 attribute = 8 bytes

1 byte per Boolean x 2 attributes = 2 bytes

Total data stored for 1 pupil = 346 bytes

There are a maximum of 200 competitors that will be entered into the competition. So therefore the maximum data stored will be 69200 bytes = 69.2 kilobytes.

The usernames also need to be stored and there won’t be more than 5 users at any one time as the number of people who should have access to the data is very small.

32 bytes per string x 5 attributes = 160 bytes

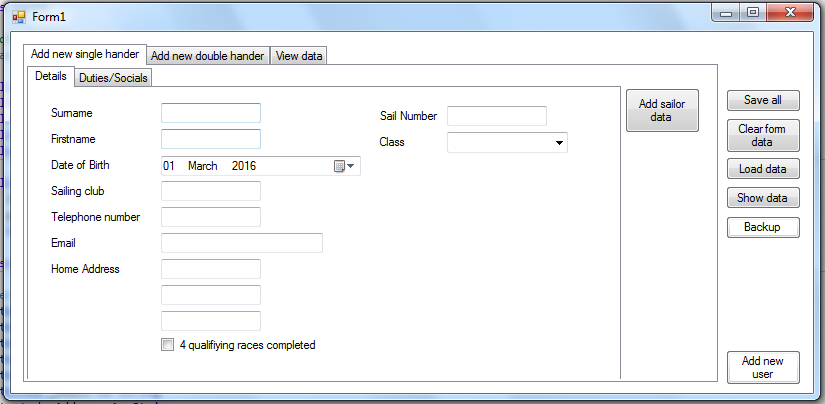
1 byte per Boolean x 1 attributes = 1 byte

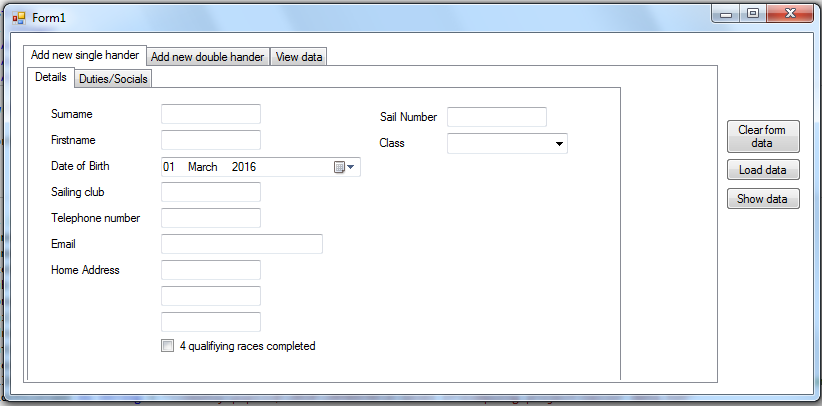
Total data stored for 1 user = 161 bytes

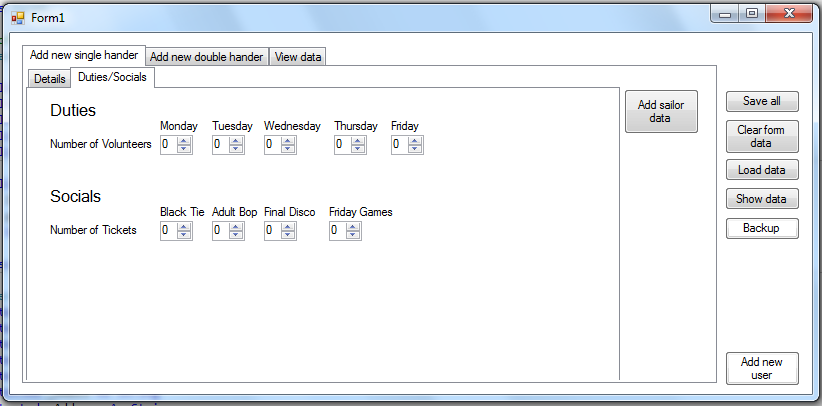
Total data stored for 5 users = 805 bytes

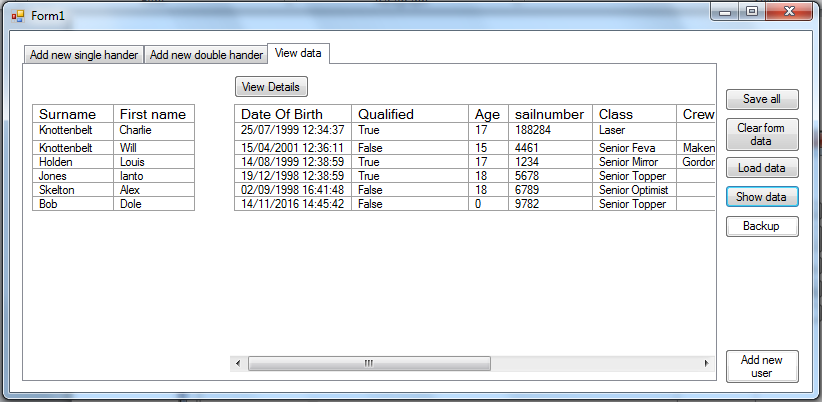
The maximum total data will be approximately 70 kilobytes to store all the data for the pupils and the users.

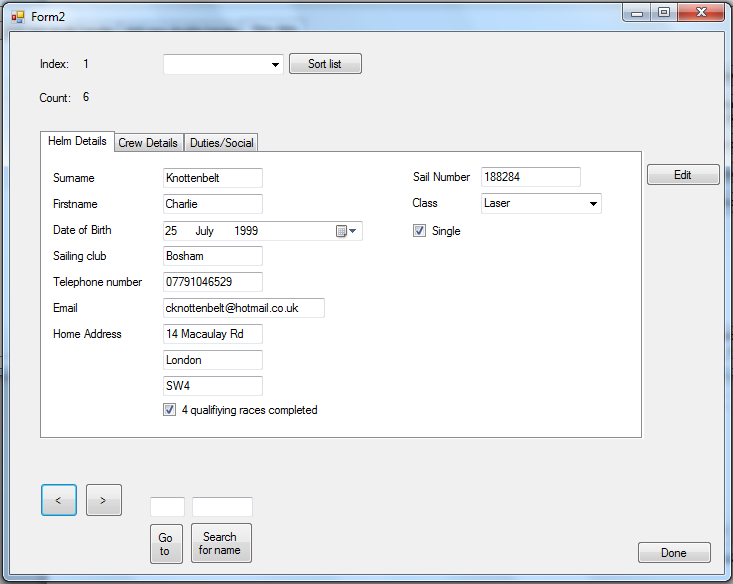
Structure



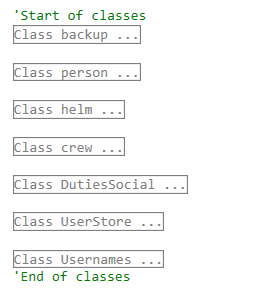


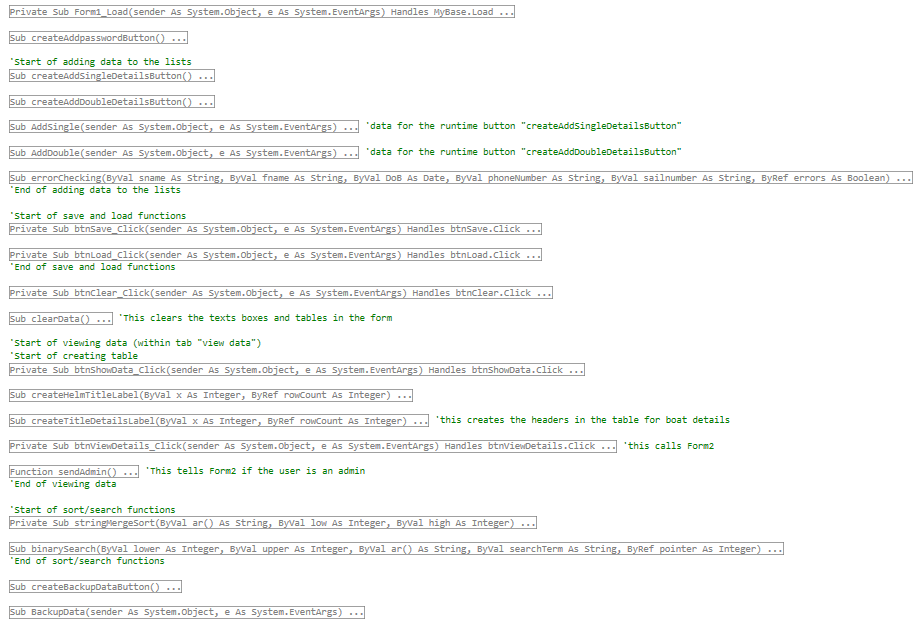




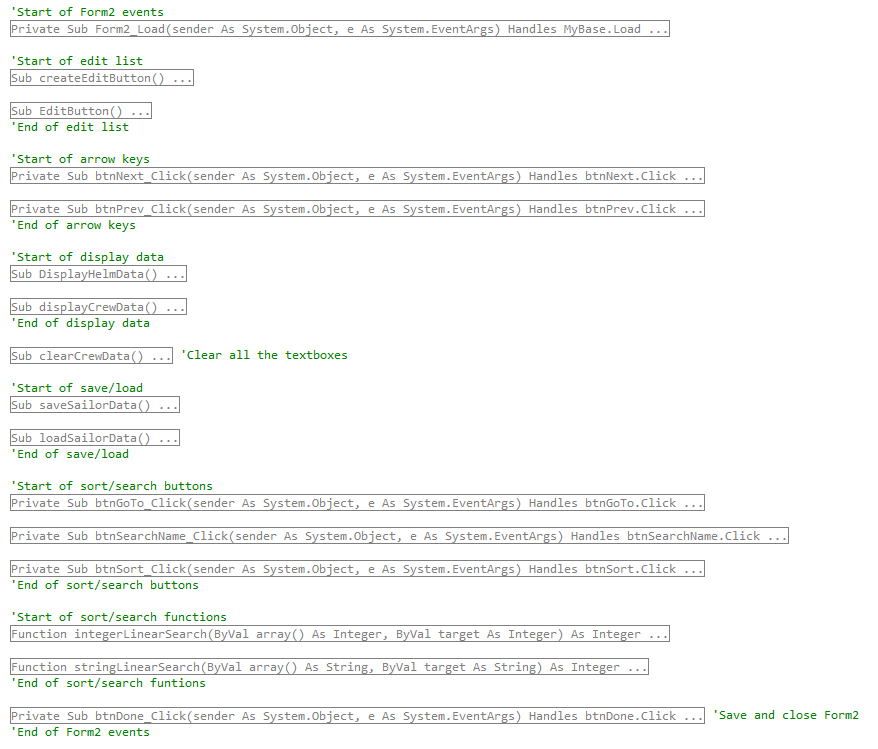


Form 1:





Form 2:



Complex algorithms:

Recursive binary search:

Binarysearch (M, low, high) {

If low <= high then

Mid = (low + high)/2

If M[mid] = Target Then return mid

If M[mid] < Target Then result = binarysearch(M, mid + 1, high)

If M[mid] > Target Then result = binarysearch(M, low, mid – 1)

Else

Return -1

End If

Return result }

Merge Sort:

MergeSort(M) {

If Len(M) <= 1 Then Return M

Left = MergeSort(FirstHalfOf(M))

Right = MergeSort(SecondHalfOf(M))

Return Merge(Left, Right) }

Merge is a function that merges 2 arrays

Security and integrity of data

Security: There will be a login system that allows admin users to edit the data however if they are not an admin they will only be able to see the data and not edit it or enter false data. This means that the data cannot be falsified and as the office that the computer that will store the data in is locked it will be very hard for unauthorised people to get access to even view the data. There also will be a backup to a raspberry pi that can store the data if there is a hardware failure on the main computer that stores this data.

Integrity: There are several validation checks that were discussed earlier to make sure that the data is integral. These include things like making sure that there were no numbers in names and that the phone number was in the correct format.

Test Strategy

To test my program, I will use several different types of testing. I will check my validation methods using white box testing and my binary search and merge sort using black box testing.

White box testing: I plan to test my binary search and merge sort with several different ranges and values to ensure that they work.

Black box testing: I will test my validation rules using white box testing so that I can ensure that all the potential errors have been covered.

Trace table: I will trace my merge sorts and binary as well as doing the white box testing to ensure that they work as planned even.

Testing

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **TestID** | **ReqID** | **Description** | **Test data / Action** | **Expected Result** | **Actual result** | **Pass** |
| T01 | R01/R04 | Erroneous - surname contains a number | Char1e | Error Message | Same | Yes |
| T02 | R01/R04 | Erroneous - date is in the future | 24/05/2023 | Error Message | Same | Yes |
| T03 | R01/R04 | Erroneous - Phone number is greater than 11 digits | 1.23457E+14 | Error Message | Same | Yes |
| T04 | R01/R04 | Erroneous - Phone number is less than 11 digits | 123 | Error Message | Same | Yes |
| T05 | R01/R04 | Erroneous - Phone number contains characters | 124r2e353y3 | Error Message | Same | Yes |
| T06 | R01/R04 | Normal - Phone number is 11 digits | 12345678910 | Accepted | Same | Yes |
| T07 | R01/R04 | Erroneous - Sailnumber contains characters | A412 | Error Message | Same | Yes |
| T08 | R02 | Store the helm data | Click Add Sailor Data button in Single tab | Data will be stored in HelmList | Same | Yes |
| T09 | R03 | Store the crew data linking it to the helm | Click Add Sailor Data button in Double tab | Data will be stored in CrewList | Same | Yes |
| T10 | R05 | Output the data to a table | Click on the show data button | Data will be output | Same | Yes |
| T11 | R06 | Output all the data to a separate form | Click on the view data button in the view details tab | A new form will open with the data in it | Same | Yes |
| T12 | R07 | Sort by surname | Select surname and click the sort button | The data will be sorted alphabeticaly by surname | Same | Yes |
| T13 | R07 | Sort by boat class | Select class and click the sort button | The data will be sorted according to seniority of class | Same | Yes |
| T14 | R07 | Normal - Search for a surname in the list | Search for a name in the list | The data for the target will be shown | Same | Yes |
| T15 | R07 | Errorneous - Search for a surname | Search for a name not in the list | Error Message | Same | Yes |
| T16 | R08 | Save data | Click save data button | The data will be saved | Same | Yes |
| T17 | R09 | Load data | Click load data button | The data will be accurately loaded according to the last save | Same | Yes |
| T18 | R10 | Normal - Login to the system as an admin | Username: "ianto" Password: "jones" | The user will be logged in as an admin | Same | Yes |
| T19 | R10 | Errorneous - Login to system with incorrect details | Username: "Felination" Password: "manbearpig" | The user will be rejected from the system | User is entered into the system as a non-admin | This is not a huge problem |
| T20 | R10 | Normal - User enters as a normal user | User does not enter any data into the username and password | User will enter as a normal user | Same | Yes |

|  |  |
| --- | --- |
| **TestID** | **Evidence** |
| T01 |  |
| T02 |  |
| T03 |  |
| T04 |  |
| T05 |  |
| T06 | No Errors |
| T07 |  |
| T08 |  |
| T09 |  |
| T10 |  |
| T11 |  |
| T12 |  |
| T13 |  |
| T14 |  |
| T15 |  |
| T16 |  |
| T17 |  |
| T18 |  |
| T19 |  |
| T20 |  |

Binary Search:

Function Binsearch(M, low, high, target)

Dim mid As Integer

Dim result As String

If low <= high Then

mid = (low + high) / 2

If M(mid) = target Then Return mid

If M(mid) < target Then result = Binsearch(M, mid + 1, high, target)

If M(mid) > target Then result = Binsearch(M, low, mid - 1, target)

Else

Return -1

End If

Return result

End Function

M = [“a”, ”c”, ”d”, ”e”, ”f”] target = “c” binsearch (M, 0, len(m) -1, “c”)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Call Number | Low | High | Mid | Return |
| 1 | 0 | 4 | 2 |  |
| 2 | 0 | 1 | 0 |  |
| 3 | 1 | 1 | 1 | 1 |
| 2 | 0 | 1 | 0 | 1 |
| 1 | 0 | 4 | 2 | 1 |

M = [“a”, ”c”, ”d”, ”e”, ”f”] target = “g” binsearch (M, 0, len(m) -1)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Call Number | Low | High | Mid | Return |
| 1 | 0 | 4 | 2 |  |
| 2 | 3 | 4 | 3 |  |
| 3 | 4 | 4 | 4 |  |
| 4 | 5 | 4 | 4 | -1 |
| 3 | 4 | 4 | 4 | -1 |
| 2 | 3 | 4 | 3 | -1 |
| 1 | 0 | 4 | 2 | -1 |

M = [“a”, ”c”, ”d”, ”e”, ”f”] target = “b” binsearch (M, 0, len(m) -1)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Call Number | Low | High | Mid | Return |
| 1 | 0 | 4 | 2 |  |
| 2 | 0 | 1 | 0 |  |
| 3 | 1 | 1 | 1 |  |
| 4 | 1 | 0 | 0 | -1 |
| 3 | 1 | 1 | 1 | -1 |
| 2 | 0 | 1 | 0 | -1 |
| 1 | 0 | 4 | 2 | -1 |

Merge Sort:

Function mergesort(ByVal ar() As Integer)

If ar.Length <= 1 Then Return ar

Dim left() As Integer = mergesort(ar.Take(ar.Length / 2).ToArray)

Dim right() As Integer = mergesort(ar.Skip(ar.Length / 2).Take(ar.Length - (ar.Length / 2)).ToArray)

Return merge(left, right)

End Function

Function merge(ByVal left() As Integer, ByVal right() As Integer)

Dim temp(left.Length + right.Length - 1) As Integer

Dim countL As Integer = 0

Dim countR As Integer = 0

Dim countT As Integer = 0

While countT < temp.Length

If countL < left.Length And countR < right.Length Then

If left(countL) < right(countR) Then

temp(countT) = left(countL)

countL += 1

countT += 1

ElseIf left(countL) > right(countR) Then

temp(countT) = right(countR)

countR += 1

countT += 1

Else

temp(countT) = right(countR)

countR += 1

countT += 1

temp(countT) = left(countL)

countL += 1

countT += 1

End If

ElseIf countL = left.Length Then

For i = countR To right.Length - 1

temp(countT) = right(i)

countT += 1

Next

ElseIf countR = right.Length Then

For i = countL To left.Length - 1

temp(countT) = left(i)

countT += 1

Next

End If

End While

Return temp

End Function

Ar=[“d”, ”c”, ”f”, ”a”, ”e”]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Call Number | Ar | Left | Right | Return |
| 1 | [“d”, ”c”, ”f”, ”a”, ”e”] | [] | [] |  |
| 2 | [“d”,”c”] | [] | [] |  |
| 3 | [“d”] | [] | [] | [“d”] |
| 2 | [“d, “c”] | [“d”] | [] |  |
| 4 | [“c”] | [] | [] | [“c”] |
| 2 | [“d”, “c”] | [“d”] | [“c”] | [“c”, “d”] |
| 1 | [“d”, ”c”, ”f”, ”a”, ”e”] | [“c”, “d”] | [] |  |
| 5 | [”f”, ”a”, ”e”] | [] | [] |  |
| 6 | [“f”] | [] | [] | [“f”] |
| 5 | [”f”, ”a”, ”e”] | [“f”] | [] |  |
| 7 | [“a”, “e”] | [] | [] |  |
| 8 | [“a”] | [] | [] | [“a”] |
| 7 | [“a”, “e”] | [“a”] | [] |  |
| 9 | [“e”] | [] | [] | [“e”] |
| 7 | [“a”, “e”] | [“a”] | [“e”] | [“a”, “e”] |
| 5 | [”f”, ”a”, ”e”] | [“f”] | [“a”, “e”] | [“a”, ”e”, “f”] |
| 1 | [“d”, ”c”, ”f”, ”a”, ”e”] | [“c”, “d”] | [“a”, ”e”, “f”] | [“a”, ”c”, ”d”, ”e”, ”f”] |