Student name: Maximilian Mihoc

Student number: C12728559

Date: 29/04/2013

*“The Technological University of Dublin (TU Dublin) will be brought into being by the convergence of Dublin Institute of Technology (DIT), Institute of Technology Tallaght (ITT) and Institute of Technology Blanchardstown (ITB) who already share this vision for higher education and have an agreed strategy for its implementation. Together the three partners will use their collective experiences, capabilities and resources to deliver higher education in a more efficient and equitable manner. The new TU Dublin will be a unitary institution formed by the three partners seizing this unique opportunity to make real change. It will be a three campus autonomous degree awarding university with critical mass, a strong online presence, and deeply embedded in all aspects of Dublin life.”*

***Part 1:***

Assuming that every institution has his own students already sorted alphabetically by name. To put all students from all three institutions together I will use a merge algorithm that will put them together and in alphabetical order in the same time. There is a way in doing that simply by join the three lists together and after that just sort that array alphabetically. This way require a quite more steps that the other one where the students are merged and sorted in the same time.

If I still want to make sure that the students from every institute are already sorted alphabetically, I will probably use the bubble sort algorithm or the insertion sort algorithm because the best case scenario for both has a big O of n, and I am sure that students are already sorted in their institutions.

**a)** The algorithms that I have considered for sorting are the bubble sort and insertion sort.

**The bubble sort** algorithm would be great just for checking if the students are in an alphabetic order in their institutions or not. This is the best sorting algorithm for doing that because is very fast and a big O of n in the best case scenario. But if I want to sort the students, if they are not in alphabetical order, the bubble sort algorithm is to slow.

**The insertion sort** algorithm is very good for both options, for checking if they are in alphabetic order (this having a Big O of N for the best case scenario) and for sorting them. The big O of this algorithm in best case scenario is Big O of n and the sorting with this is much faster than bubble sort. With bubble sort if the students are not in order you have to swap them every time when one is after another and should be before, too slow and to many swaps, but with insertion sort, if a student is not where he should be, the algorithm will find a good place for him in order, and will put that student there, changing the other students places.

**b)** The best data structure that will fit in this case is a linked list because in an institution are many students and if they are not in an alphabetic order, every time when you have to change a student place in list, just change the pointer value in a good way and you don't care about everything else. Using an array in this case, every time when you find a student in a wrong place, you have to move all students after his place in list to the right with one position, and fit this student in his good place.

Using an array to sort all students from one institute isn't probably the best idea and for this, a linked list is better.

**c)** Even if I said a linked list is better than an array, this doesn't make a difference if the number of students is small and known. For my solution of this problem, I used only 5 students from DIT, 4 students from ITT and 3 students from ITB.

I know how many students are in every institution, I know they are already in alphabetical order in their institution and I know that for this case, it is ok to use linked list and arrays as well. I will chose an array this time.

Students from every institution are sorted in alphabetic order by name.

I created a record with Student number, Student name, Year, actual college and previous college, (if the student has one) for knowing the student details.

To hold information of every students from all institutions, I created three arrays of structs. One for DIT students, one for ITT students and one for ITB students.

A fourth array that I created is the array of TU students that have to hold all students from every institution sorted alphabetically by name.

In my algorithm, I used a loop to go through the three arrays in which I have the students in their own institution. Starting with the first student from every array, I created a code to compare their name and put them in order. If they have the same name, the algorithm will put them in a random order in the TU array. If one of the array is finished, this loop will stop and the other 2 arrays are checked in another loop, which student should be before and which after, until another array is finished. The last array that isn't finished yet, will be just copied in the order that it is already because is sorted and we don't have another elements to compare with.

When the program is finished, the TU array will contain all students from DIT, ITT and ITB institution.

In my algorithm I have 7 while loops but for any values in those arrays, doesn't meter how many elements they have, this algorithm will execute just three loops.

**d)**

l1 -> length of DITs array

l2 -> length of ITTs array

l3 -> length of ITBs array

l4 -> length of TUs array

In my code, I am using the less than "<" sign to compare strings. This condition: **if (string1 < string2)** means if the string 1 should be before string 2 in alphabetic order.

**Pseudo-code Algorith**

1: **Struct** student\_rec

2: studentNumber; name; year; College; previousCollege;

3: END STRUCT;

4: Initialise the arrays with student details;

5: Struct student\_rec DITs[10], ITTs[10], ITBs[10], TUs[30];

6: i=0; j=0; m=0; k=-1;

7: WHILE ( i<= l1 and j<= l2 and m <= l3) DO

8: IF( DITs[i].name < ITTs[j].name)

9: then IF ( DITs[i].name < ITBs[m].name)

10: then k=k+1;

11: TUs[k] = DITs[i];

12: i++;

13: ELSE k = k + 1;

14: TUs[k] = ITBs[m];

15: m++;

16: ELSE IF ( ITTs[j].name < ITBs[m].name )

17: THEN k = k + 1;

18: TUs[k] = ITTs[j]

19: j++;

20: ELSE k = k + 1;

21: TUs[k] = ITBs[m];

22: m ++;

23: END WHILE;

24: WHILE ( i <= l1 and j <= l2 ) DO

25: IF( DITs[i].name < ITTs[j].name) then

26: k=k+1;

27: TUs[k] = DITs[i];

28: i++;

29: ELSE k=k+1;

30: TUs[k] = ITTs[j];

31: j++;

32: END WHILE;

33: WHILE ( i <= l1 and m <= l3 ) DO

34: IF( DITs[i].name < ITBs[m].name) then

35: k=k+1;

36: TUs[k] = DITs[i];

37: i++;

38: ELSE k=k+1;

39: TUs[k] = ITBs[m];

40: m ++;

41: END WHILE;

42: WHILE ( j <= l2 and m <= l3 ) DO

43: IF( ITTs[j].name < ITBs[m].name) then

44: k=k+1;

45: TUs[k] = ITTs[j];

46: j++;

47: ELSE k=k+1;

48: TUs[k] = ITBs[m];

49: m ++;

50: END WHILE;

51: WHILE ( i <= l1) DO

52: k=k+1;

53: TUs[k] = DITs[i];

54: i++;

55: END WHILE;

56:WHILE ( j <= l2) DO

57: k=k+1;

58: TUs[k] = ITTs[j];

59: j++;

60: END WHILE;

61: WHILE ( m <= l3) DO

62: k=k+1;

63: TUs[k] = ITBs[m];

64: m++;

65: END WHILE;

END.

**Step by step description:**

In first sis lines I initialise and declare all variables and arrays

In line 7, check if I have elements in all three arrays from all three institutions. The variables i, j, and m are counters for every array. If one of them are equal to length of their array this means we don't have any elements in that array and we need to get out from that while.

When the condition from while is ok, I am doing the minimum from three elements in arrays: element 0 from array 1, element 0 from array2 and element 0 for array 3. The minimum from those, means the name from all three arrays that should be first in the forth array, will be added in the forth array. For example if the element from ITTs array should be first, if it is the smallest one, then add that element in TUs array, increase the length of TU array, increase the counter for array 2 (go to the next element in array 2 <j=j+1>). Same thing will be applied if the smallest element is in DIT array or in ITB array. The algorithm will do that until one of the array is finished. All this things are made in lines 8 - 22.

After one array is finished, the while loop is finished.

If the ITB array will be finished first, the algorithm will execute lines 24 - 32 and will merge in alphabetic order the elements from DIT array and ITT array until one of them will be finished as well.

If the ITT array will be finished first, the algorithm will execute lines 33 - 41 and will merge in alphabetic order the elements from DIT array and ITB array until one of them will be finished.

If the DIT array will be finished first, the algorithm will execute lines 42 - 50 and will merge in alphabetic order the elements from ITT array and ITB array until one of them will be finished.

After all this, the algorithm will reach the line 51 just with one array that isn't finished yet.

If that array is the DIT array, the algorithm will execute the lines number 51- 55. On these lines, all the remaining elements from DIT will be added in the TU array.

If that array is the ITT array, the algorithm will execute the lines number 56 - 60. On these lines, all the remaining elements from ITT will be added in the TU array.

If that array is the ITB array, the algorithm will execute the lines number 61 - 65. On these lines, all the remaining elements from ITB will be added in the TU array.

In the end, the TU array will contain all elements from DIT, ITT and ITB in alphabetic order

**Flowchart**



**Part 2**

Explain two good ways of searching for a student who was in IT Tallaght and is now in DIT.

**a)**  I was thinking to use a **binary search** to search in my TU array a student that was in ITT institute and is now in DIT but this won't work in my case because in my array, the students are sorted alphabetically by name, and for this search the array needs to be sorted by college name or by previous college name since this search is looking for a student that was in Tallaght and in now in DIT. The most important rule for a binary search is: The array should be sorted, and this sort should match the search key.

If the array in that this algorithm is applied is sorted, this algorithm require just few steps to find the search key. In my array, I can apply a binary search just if I want to find a student by name. The algorithm will find the middle of my array, check if the name should be in left or right side of middle, go in that side, do the same until the search element is found.

If I would know the name of that student which was in ITT and is now in DIT, I could apply a binary search and the algorithm could be more efficient.

A **linear search** require more steps than a binary search but for this search, there is no need that an array should be sorted. My goal in this search is to find a student which has the previous college ITT and the current college DIT.

The steps in this search algorithm are:

- go to the first student, check if the previous college is ITT and current college is DIT, if it is, display the name of this student and stop the search, if not, go to the next student check again if the previous college is ITT and current college is DIT. Do the same for all students until you find that student.

The best search algorithm for my program is the linear search because the array should not be sorted by college and this will definitely find that student. The binary search isn't the best idea because, if I want to apply this, i should sort the array first and after that apply this algorithm.

b) **Pseudo-Code**

In my algorithm, I will find that student that was in ITT and is now in DIT and return his name.

PROGRAM search student:

FOR i=0; i<=l4 DO

IF ( TUs[i].previousCollege = ITT and TUs[i].college = DIT )

THEN return TUs[i].name;

END FOR;

END.

**Flowchart**

