**ASSIGNMENT 1 FRONT SHEET**

|  |  |  |  |
| --- | --- | --- | --- |
| **Qualification** | **BTEC Level 5 HND Diploma in Computing** | | |
| **Unit number and title** | Unit 19: Data Structures and Algorithms | | |
| **Submission date** | 8/10/2022 | **Date Received 1st submission** |  |
| **Re-submission Date** |  | **Date Received 2nd submission** |  |
| **Student Name** | Do Huu Duy | **Student ID** | GCC200018 |
| **Class** | GCC0903 | **Assessor name** | Tran Thi Kim Khanh |
| **Student declaration**  I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice. | | | |
|  |  | **Student’s signature** | huuduy |

**Grading grid**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **P1** | **P2** | **P3** | **M1** | **M2** | **M3** | **D1** | **D2** |
|  |  |  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **❒ Summative Feedback: ❒ Resubmission Feedback:** | | |
| **Grade:** | **Assessor Signature:** | **Date:** |
| **Internal Verifier’s Comments:** | | |
| **IV Signature:** | | |

# Assignment Brief 1 (RQF)

## Higher National Certificate/Diploma in Business

|  |  |
| --- | --- |
| **Student Name/ID Number:** |  |
| **Unit Number and Title:** | Unit 19: Data Structures and Algorithms |
| **Academic Year:** | **2021** |
| **Unit Assessor:** |  |
| **Assignment Title:** | Examine and specify ADT and DSA |
| **Issue Date:** |  |
| **Submission Date:** |  |
| **Internal Verifier Name:** |  |
| **Date:** |  |

|  |
| --- |
| **Submission Format:** |
| Format:   * The submission is in the form of an individual written report and a presentation. This should be written in a concise, formal business style using single spacing and font size 12. You are required to make use of headings, paragraphs and subsections as appropriate, and all work must be supported with research and referenced using the Harvard referencing system. Please also provide a bibliography using the Harvard referencing system.   Submission   * Students are compulsory to submit the assignment in due date and in a way requested by the Tutor. * The form of submission will be a soft copy posted on <http://cms.greenwich.edu.vn/>. * Remember to convert the word file into PDF file before the submission on CMS.   Note:   * The individual Assignment *must* be your own work, and not copied by or from another student. * If you use ideas, quotes or data (such as diagrams) from books, journals or other sources, you must reference your sources, using the Harvard style. * Make sure that you understand and follow the guidelines to avoid plagiarism. Failure to comply this requirement will result in a failed assignment. |
| **Unit Learning Outcomes:** |
| **LO1** Examine abstract data types, concrete data structures and algorithms  **LO2** Specify abstract data types and algorithms in a formal notation |
| **Assignment Brief and Guidance:** |
| **Assignment scenario**  You work as in-house software developer for Softnet Development Ltd, a software body-shop providing network provisioning solutions. Your company is part of a collaborative service provisioning development project and your company has won the contract to design and develop a middleware solution that will interface at the front-end to multiple computer provisioning interfaces including SOAP, HTTP, JML and CLI, and the back-end telecom provisioning network via CLI.  Your account manager has assigned you a special role that is to inform your team about designing and implementing abstract data types. You have been asked to create a presentation for all collaborating partners on how ADTs can be utilized to improve software design, development and testing. Further, you have been asked to write an introductory report for distribution to all partners on how to specify abstract data types and algorithms in a formal notation.  **Tasks**  **Part 1**  You will need to prepare a presentation on how to create a design specification for data structures, explaining the valid operations that can be carried out on the structures using the example of:   1. A stack ADT, a concrete data structure for a First In First out (FIFO) queue. 2. Two sorting algorithms. 3. Two network shortest path algorithms.   **Part 2**  You will need to provide a formal written report that includes the following:   1. Explanation on how to specify an abstract data type using the example of software stack. 2. Explanation of the advantages of encapsulation and information hiding when using an ADT. 3. Discussion of imperative ADTs with regard to object orientation. |

|  |  |  |
| --- | --- | --- |
| Learning Outcomes and Assessment Criteria (Assignment 1) | | |
| Pass | Merit | Distinction |
| **LO1** Examine abstract data types, concrete data structures and algorithms | | **D1** Analyse the operation, using illustrations, of two network shortest path algorithms, providing an example of each. |
| **P1** Create a design specification for data structures explaining the valid operations that can be carried out on the structures.  **P2** Determine the operations of a memory stack and how it is used to implement function calls in a computer. | **M1** Illustrate, with an example, a concrete data structure for a First In First out (FIFO) queue.  **M2** Compare the performance of two sorting algorithms. |
| **LO2** Specify abstract data types and algorithms in a formal notation | | **D2** Discuss the view that imperative ADTs are a basis for object orientation and, with justification, state whether you agree. |
| **P3** Using an imperative definition, specify the abstract data type for a software stack. | **M3** Examine the advantages of encapsulation and information hiding when using an ADT. |

Table of Contents

[I. Create a design specification for data structures explaining the valid operations that can be carried out on the structures 8](#_Toc116132966)

[1. What is Abstract Data Structure 8](#_Toc116132967)

[2. Five steps to implement an Abstract Data Structure (ADT) 8](#_Toc116132968)

[3. Example about Singly Linked List 9](#_Toc116132969)

[II. Determine the operations of a memory stack and how it is used to implement function calls in a computer 12](#_Toc116132970)

[1. What is Stack 12](#_Toc116132971)

[2. How a memory stack work 13](#_Toc116132972)

[3. What is a function call in a computer and how does it work 14](#_Toc116132973)

[4. How can stack work in function calls 16](#_Toc116132974)

[III. Using an imperative definition, specify the abstract data type for a software stack 18](#_Toc116132975)

[1. Explain why I choose describing the SinglyLinkedList ADT 18](#_Toc116132976)

[2. What is an imperative ADT 19](#_Toc116132977)

[3. Using an imperative definition, specify my chosen ADT 19](#_Toc116132978)

[4. Explain the problem 21](#_Toc116132979)

[5. Show what effect does the ADT obtain after an operation is called 21](#_Toc116132980)

[IV. Illustrate, with an example, a concrete data structure for a First In First out (FIFO) queue 33](#_Toc116132981)

[V. Compare the performance of two sorting algorithms 36](#_Toc116132982)

[1. Sorting algorithms 36](#_Toc116132983)

[2. Bubble sort 37](#_Toc116132984)

[3. Quick sort 39](#_Toc116132985)

[VI. Examine the advantages of encapsulation and information hiding when using an ADT 42](#_Toc116132986)

[1. What is Encapsulation and Information Hiding 42](#_Toc116132987)

[2. The differences between encapsulation and information hiding 42](#_Toc116132988)

[3. What do I use to represent an ADT and why 43](#_Toc116132989)

[4. When using an ADT, what is the role of Encapsulation and Information Hiding 43](#_Toc116132990)

[References 44](#_Toc116132991)

# Table of Pictures

[Figure 1. Five step to implement ADT 8](#_Toc116151027)

[Figure 2. Add first model 9](#_Toc116151028)

[Figure 3. Add last model 10](#_Toc116151029)

[Figure 4. Get first model 10](#_Toc116151030)

[Figure 5. Get last model 10](#_Toc116151031)

[Figure 6. Remove first model 11](#_Toc116151032)

[Figure 7. Remove last model 12](#_Toc116151033)

[Figure 8. Memory stack work 14](#_Toc116151034)

[Figure 9. Function call work 15](#_Toc116151035)

[Figure 10. Code function call 16](#_Toc116151036)

[Figure 11. Push method to stack model 17](#_Toc116151037)

[Figure 12. Pop method out of stack model 18](#_Toc116151038)

[Figure 13. Example about queue 33](#_Toc116151039)

[Figure 14. Code random of shuffle method 34](#_Toc116151040)

[Figure 15. Code of queue about the problem buy the movie ticket with discount cost 35](#_Toc116151041)

[Figure 16. The first five element of queue 35](#_Toc116151042)

[Figure 17. Get the first element of queue 35](#_Toc116151043)

[Figure 18. Ge the second element of queue 35](#_Toc116151044)

[Figure 19. Ge the third element of queue 35](#_Toc116151045)

[Figure 20. Ge the fourth element of queue 36](#_Toc116151046)

[Figure 21. Ge the fifth element of queue 36](#_Toc116151047)

[Figure 22. Result console application 36](#_Toc116151048)

[Figure 23. Code bubble sort 37](#_Toc116151049)

[Figure 24. The unsorted array 37](#_Toc116151050)

[Figure 25. The array is sorting by compare the two numbers 1 and 3 37](#_Toc116151051)

[Figure 26. The array is sorting by compare the two numbers 3 and 2 38](#_Toc116151052)

[Figure 27. The array is sorting by compare the two numbers 3 and 4 38](#_Toc116151053)

[Figure 28. The array is sorting by compare the two numbers 4 and 6 38](#_Toc116151054)

[Figure 29. The array is sorting by compare the two numbers 6 and 5 38](#_Toc116151055)

[Figure 30. Result of the array after sorting 38](#_Toc116151056)

[Figure 31. The calculation of the best case and average case of the bubble sort 39](#_Toc116151057)

[Figure 32. Code of quick sort 39](#_Toc116151058)

[Figure 33. Code of quick sort 40](#_Toc116151059)

[Figure 34. Example about Quick Sort 41](#_Toc116151060)

# I. Create a design specification for data structures explaining the valid operations that can be carried out on the structures

## 1. What is Abstract Data Structure

An object's behavior can be described by a set of values and a set of actions, and this behavior is known as an abstract data type (ADT). The definition of ADT merely specifies the actions that must be taken, not how they must be carried out. It is unclear what algorithms will be utilized to carry out the operations and how the data will be structured in memory. Because it provides an implementation-independent perspective, it is called "abstract." (geeksforgeeks, 2022)

## 2. Five steps to implement an Abstract Data Structure (ADT)

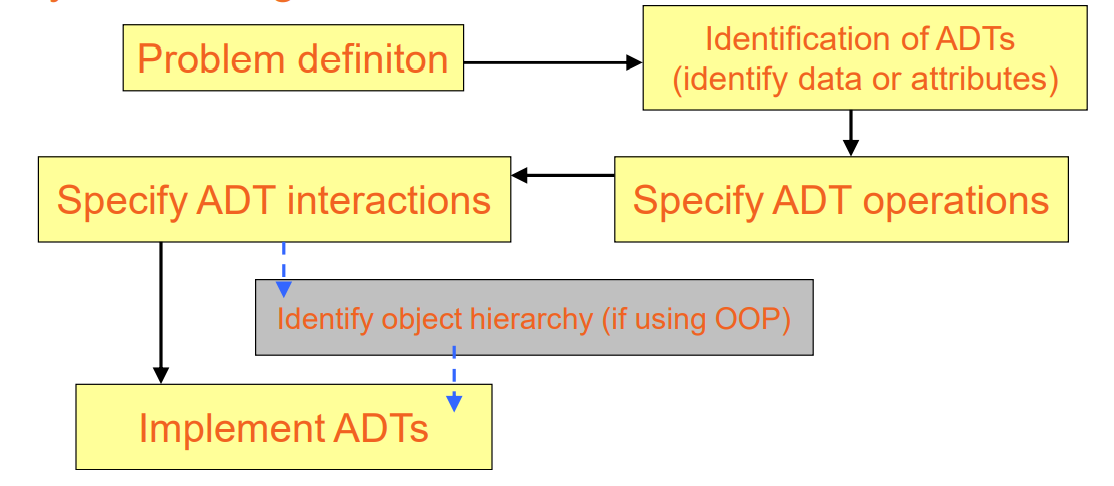


Figure 1. Five step to implement ADT

* Problem definition: We need to identify the problem and then, we will base on the problem that is identified and create an ADT to solve that problem. Then, we will describe specifical the problem. In this case, I use Singly Linked List to solve the problem about manage players.
* Identify data or attributes of ADT: In this step, we will identify the attributes or data for the ADT, based on the problem that is identified in step one. I will determine the data type used in this problem. In this case at part 5 of III, I will manage the players of a club so the data type that I use will be object type.
* Specify ADT interactions:In this step, we will determine the interactions of ADT. Determine Is-A and Has-A. For example, in this case in class players have Name, Age, and Nationality… and in this case we will use the Has-A. Example, the Singly Linked List has a Node<E>
* Specify ADT operations:Now, we need to list that ADT the operation. For example, with the Singly Linked List ADT, the ADT will operate functions such as addFirst(), addLast(), removeFirst(), removeLast(), getFirst(), and getLast()
* Implement ADT: From the 4 steps defined above, we will implement the ADT. We will create the class players have some attributes of a player, then will create Singly Linked List interface to implement the SinglyLinkedList<Player> and Node class to get pointer next or previous and head of the Singly Linked List. Last, we will create main class to run the application.

## 3. Example about Singly Linked List

A linear data structure is a singly linked list. In contrast to arrays, linked list elements are connected together using pointers rather than being stored in a continuous space. They consist of a number of interconnected nodes. Each node in this structure carries data, which can be an object, a string, an integer, but not a primitive type, as well as the address of the node after it (geeksforgeeks, 2022)

There are 8 supported operations of Singly Linked List:

* **addFirst(E element):** In this operation, I will add an element in front of the list and increases the size. Firstly, I will transmit into this function an element and then I will turn it into a node. When I turn it into a node, I will add them to the list and increase the number of elements of the list. The picture 2 is a description of this function.

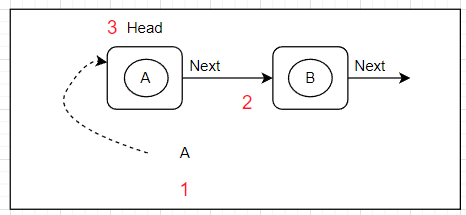


Figure 2. Add first model

1. A is an element
2. Set the next pointer of node A to node B
3. Set node A to head

* **addLast(E element):** In this operation, I will add an element after the last element of the list and increases the size. Firstly, I will transmit into this function an element and then I will turn it into a node. Next, I have to check whether the list has a node or not. If the list doesn't have any node, I will add the node just create to the list and increase the size of the list. Besides, when the list has the node in it, I will create a temp variable to save the current head node in the list, then I will use the while loop to check whether the next pointer of the current head node is a different null or not. If the next pointer of the current head node is not null, I will set the head position of the list for the next node but if the next pointer of the current head node is null, I will add the node just create to the list and increase the size of the list. The picture 3 is a description of this function.

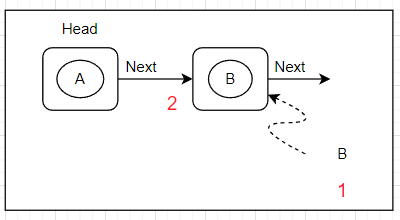


Figure 3. Add last model

1. B is an element
2. Set the next pointer of node A to node B

* **E getFirst():** In this operation, I will get data from the first element of the list. In order to do this operation, I need to check whether the list has elements or not firstly. If the list has the element, I will get data of the first element and display them. Besides, if the list has no element, I will give an exception to notify the list is empty. The picture 4 is a description of this function.

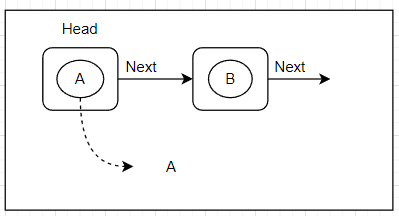


Figure 4. Get first model

* **E getLast():** In this operation, I will get data from the last element of the list. In order to do this operation, I need to check whether the list has elements or not firstly. If the list has the element, I will get data of the last element and display them. Besides, if the list has no element, I will give an exception to notify the list is empty. The picture 5 is a description of this function.

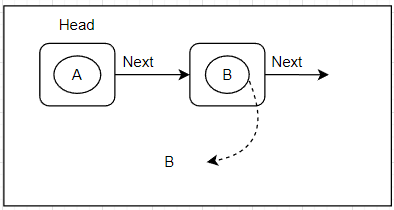


Figure 5. Get last model

* **E removeFirst():** In this operation, I will remove the first element of the list and return that element, then decrease the size of the list. In order to do this operation, I need to check whether the list has elements or not firstly. If the list has the element, I will create two temp variables to save the first element of the list and the element next to the first element of the list. Then, I will remove the first element of the list by set the next pointer of the first element of the list to null. Next, I set the element next to the first element of the list to head position, and then return the element removed and decrease the size of the list. Besides, if the list has no element, I will give an exception to notify the list is empty. The picture 6 is a description of this function.

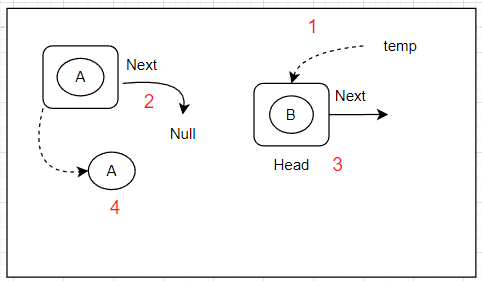


Figure 6. Remove first model

1. Create a temp variable to save the Node B
2. Set the next pointer of node A to null
3. Set Node B to head
4. Return node A

* **E removeLast():** In this operation, I will remove the last element of the list and return that element, then decrease the size of the list. In order to do this operation, I need to check whether the list has elements or not firstly. If the list has the element, I will create two temp variables to save the last element of the list and the element previous to the last element of the list. Then, I will remove the last element of the list by setting the next pointer of the front element the last element to null and returning the last element, and decreasing the size of the list. Besides, if the list has no element, we will give an exception to notify the list is empty. The picture 7 is a description of this function.

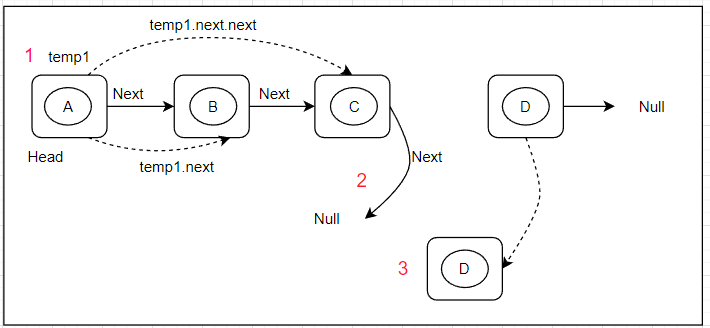
****

Figure 7. Remove last model

1. Create a temp1 variable to save the current head of the list. I will use while loop to perform temp1 variable and temp1 will run temp1.next.next if it is different null, I will set for this variable run temp1.next and continue to loop until the temp1.next.next == null I will set the temp1.next == null, now I removed the last node, but if it the temp1.next.next == null, I will set the temp1.next == null, now I removed the last node.
2. Set the next pointer of the node near the last node equal null
3. Return the last node

* **int size():** This operation, I use it to check the size of the list, and it will return the size of the list.
* **Boolean isEmpty():** This operation, I use to check whether the list has empty or not. I check if the head position of the list is different from null. If the list is different than null the function will notify a message “List is not empty” and return true. But if the list is null the function will notify a message “List is empty” and return false.

# II. Determine the operations of a memory stack and how it is used to implement function calls in a computer

## 1. What is Stack

Stack is an abstract data type that is frequently used and has the two main operations of push() and pop(). In the top element in the stack, the push() and the pop() method will perform on it. While the pop() function removes an element from the top spot of the stack, the push() function adds an element to the stack and they work in principle Last In First Out (LIFO). In a linear data structure type, a stack represents a series of items or components. All operations are done in the top position of the stack, which has a bottom that is bound. Because the stack just have an entrance and exit, due to this, the stack works in the LIFO principles. For example, when we use push() function to add an element to the stack, now it is the top of stack and then we add more an element to the stack, now the stack that we just add will be top of the stack. When we use pop() function to get the element of the stack, the element that is pushed last to the stack will be got because it is the top of stack so we will get it. The top value of the stack is always increased by one whenever a push() function is used to add an element to the stack, and it is always decreased by one whenever an element is removed from the stack by the pop() function. The term "stack pointer" refers to a pointer that points to the top of the stack (techopedia, 2022)

There are five supported operations in Stack:

* + push(E element): This operation is used to add an element at the top of the stack and increases the size.
  + E pop(): This operation is used to remove an element at the current top of the stack and returns it. If the stack is empty throw IllegalStateException with a message “Stack is empty”
  + E peek(): This operation is used to return the element at the current top of the stack and if the stack is empty throw throw IllegalStateException with a message “Stack is empty”
  + int size(): This operation is used to return the size of the stack
  + Boolean isEmpty(): This operation is used to check whether the stack is empty or not

Exception of the stack: There are two exceptions of stack, this is stack full and stack empty. When we push the element to the stack but the stack is full, the stack will throw an exception to notify stack is full. For example, when we set the max size for stack is 5, then we push first five elements to the stack, now the stack is full; but if we push an more element to the stack, the stack will throw an exception to notify stack is full. In a other hand, for the stack empty exception. When we pop the element out of the stack but in the stack do not have any element, now the stack will throw an exception to notify the stack is empty.

## 2. How a memory stack work

Stack memory is a technique for managing memory that enables system memory to be utilized as a first-in, last-out buffer for temporary data storage. The register known as the Stack Pointer is one of the crucial components of stack memory operation. Every time a stack action is executed, the stack pointer, which displays the location of the current stack memory location, is automatically changed. Using the PUSH instruction, storing data on the stack is known as pushing, and removing data from the stack is known as popping (using the POP instruction). The picture 8 is a description about the way that a memory stack work (Yiu, 2015)

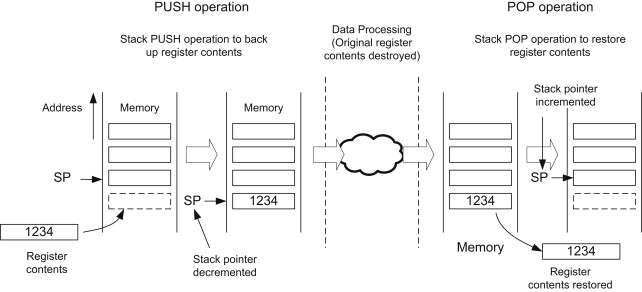


Figure 8. Memory stack work

Now, we can see that 1234 is the address of the new element, and the element is pushed to the stack and that element is saved to the memory with the address 1234. The element with the address 1234 is pushed to the stack and it is a top position in the stack. Next, the element with address 1234 is popped out of the stack by Pop operation.

## 3. What is a function call in a computer and how does it work

A method in Java is a group of statements that carry out a certain action or function. It is commonly used because it enables code reuse, which refers to the ability to write code once and use it multiple times. It also allows for simple modification. Each method has a unique name that it goes by when called. The method is called and completes the stated task when the compiler reads the method name. A description of the function call is shown in picture 10. We can see, when the application is executed the main will call to A() method and then the A() method will call the B() method and then the B() method will call the C() method. Each of the methods listed above will carry out a specific action when it is called since, as we can see, they are all static methods (javatpoint, 2021)

Each running computer program makes use of a section of memory known as the stack to make functions possible. The stack is used by the machine to store local variables, function arguments, return data, and registers for later restoration. A stack frame is the area of the stack designated for a single function call. In other words, new space (i.e., a stack frame) is created on the stack for each function call. The very top locations in memory are where the computer stack is located. According to its name, a stack data structure, a stack is a data structure with the "top" increasing from high-value addresses toward low-value addresses. Since that is the memory region connected to our current function calls, the topmost stack frame is what matters most when we discuss function calls. A description of how a function call works is shown in picture 9 (Hu, 2022)

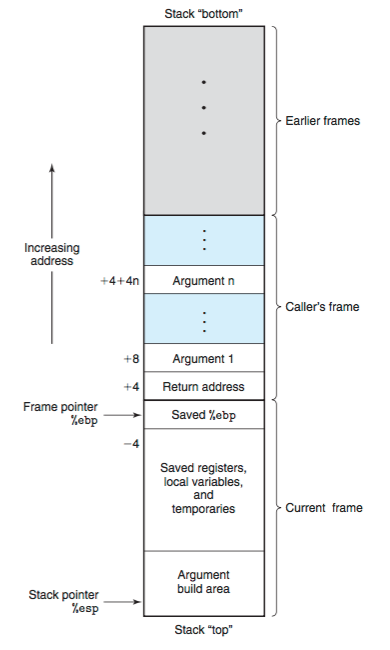


Figure 9. Function call work

## 4. How can stack work in function calls

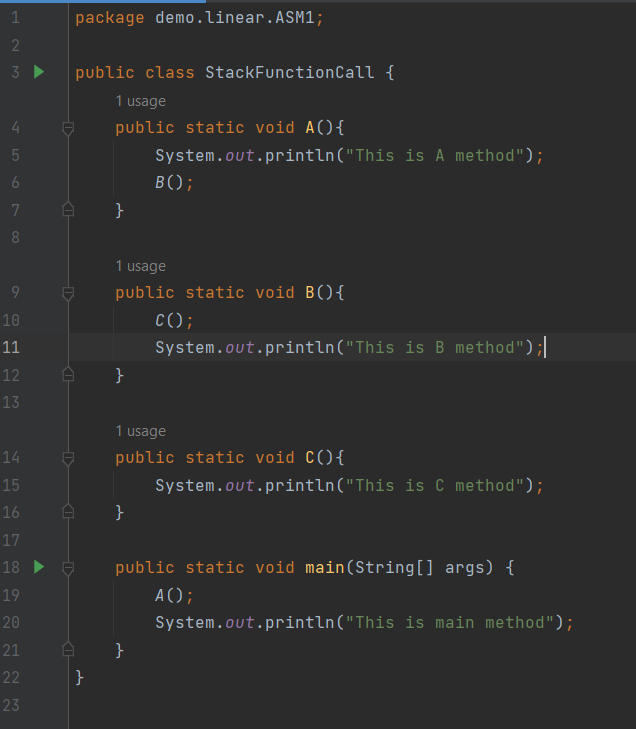


Figure 10. Code function call

With the above code, the stack works in the function calls the following. Firstly, when the application run, the main() method will be added to stack and call A() method and A() method will be added to stack and print a message "This is A method", then the A() method will call the B() method. B() method will be added to stack and the B() method will call to the C() method. C() method will be added to stack, and print a message "This is C method" and then, in the B() method will print a message "This is B method" and the last in the main() method will print a message "This is main method" and stop the system.

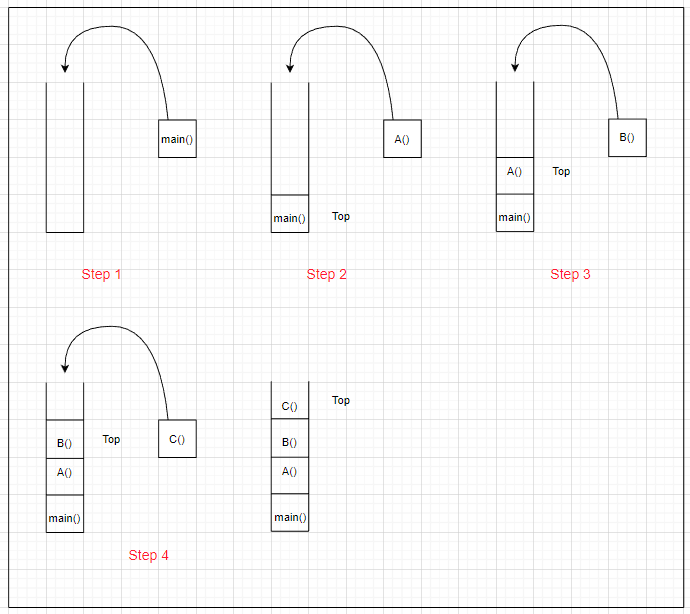


Figure 11. Push method to stack model

Based on picture 2, the main() method will be pushed to the stack in step 1. Then, the A() method will be pushed to the stack next in step 2. next, the B() method will be pushed to the stack in step 3. Last, The C() method will be pushed to the stack in step 4

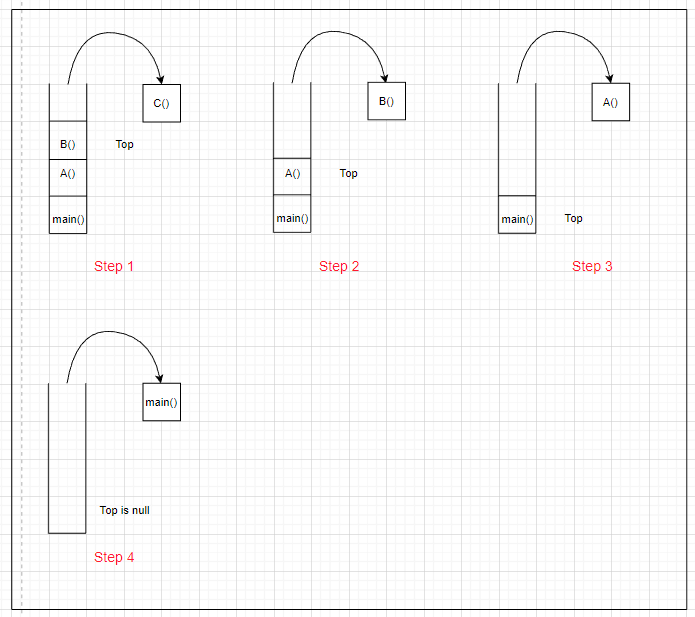


Figure 12. Pop method out of stack model

Based on picture 3, now the stack has four methods and we will perform getting each method in principle LIFO of the stack. Firstly, the C() method will be popped out of the stack. Next, the B() method will be popped out of the stack and we will perform this until the stack is empty. After popping the B() method we will continue to pop the A() method out of the stack and last is the main() method. When we pop the main() method out of the stack, now the stack is empty and finished.

# III. Using an imperative definition, specify the abstract data type for a software stack

## 1. Explain why I choose describing the SinglyLinkedList ADT

I choose Singly Linked List to describe because this is an ADT that is easy to use since it just looks in one direction. Besides, this is an ADT that uses less memory than Doubly Linked List. For the Singly Linked List, I just work on it to a pointer Next. It gives me easy-to-use and convenience when managing and checking the data in the list. I can add, remove, and get the data in the list easily.

## 2. What is an imperative ADT

In the "basic" view, which is nearer to the way of thinking of basic programming dialects, a theoretical information structure is considered as an element that is alterable — implying that it could be in various states at various times. A few tasks might impact the condition of the ADT; in this manner, the request in which tasks are assessed is significant, and similar procedures on similar elements might make various impacts whenever executed at various times — very much like the directions of a PC, or the orders and methods of a basic language. To highlight this view, it is standard to say that the tasks are executed or applied, as opposed to assessed. The basic style is in many cases utilized while depicting conceptual calculations (liquisearch, 2022)

## 3. Using an imperative definition, specify my chosen ADT

Class Node<E>:

Instace variables:

E: Elements := E

Next := Node<E>

Class SinglyLinkedList<E>:

Instance variables:

size: size of the list := int

head: head of the list := Node<E>

Operations:

* Public addFirst: E x -> ()

newNode <- Node<E>(x)

Pre:

none

Post:

size++

head <- newNode

Invariant:

size >= 0

* Public getFirst: () -> E

Pre:

size > 0

Post:

none

Invariant:

size >= 0

head do not change

* Public getLast: () -> E

oldHead <- head := Node<E>

Pre:

size > 0

Post:

A <- B.next

Invariant:

size >= 0

head do not change

* Public removeLast: () -> E

Case 1: if size = 1

oldHead <- head := Node<E>

Pre:

size > 0

Post:

head <- null

size--

Invariant:

size >= 0

head do not change

Case 2: if size != 1

oldHead <- head := Node<E>

temp := E

Pre:

size > 0

Post:

oldHead <- null

size--

Invariant:

size >= 0

head do not change

Formal notation:

E: Any data type, except primitive types

x: Value of E type

A: The last Node of Singly Linked List

B: The Node near the last Node of Singly Linked List

## 4. Explain the problem

I use Singly Linked List to manage the player of the club. The application can add a new player, remove the player from the list and I can see all information about the player in the list and all of the players in the list. In this application, I use 9 functions such as add first, add last, remove first, remove last, get first, get last, check the size of the list, check whether the list is empty or not, and show all of the players in the list.

## 5. Show what effect does the ADT obtain after an operation is called

|  |  |  |
| --- | --- | --- |
| Code in main | Actual Output | Explanation |
|  | The list initial have 2 players. | In this function, I will add the player to the first of the list. When the application is executed, I will choose 1 number to perform and add the first function. The addFirst() function of the Singly Linked List will be performed. After adding successfully the application display the size of the list as 3 and the head of Singly Linked List will be changed and the size of the list will increase to 1. |
|  | Now, the list have 3 players. | In this function, I will add the player to the last of the list. When the application is executed, I will choose 2 numbers to perform and add the last function. The addLast() function of Singly Linked List will be performed. After adding successfully the application displays the size of the list as 4 because I added the 3 player to the list before. Now the size of the Singly Linked List will be changed to 4 and the next pointer of the third node will be pointed to the node next to it. |
|  | Now, the first players of the node is Thai. | In this function, I will remove the first player from the list. When the application is executed, I will choose 3 numbers to perform and remove the first function. The removeFirst() function of the Singly Linked List will be performed. After removing successfully the application displays the player removed and display the size of the list as 3 because, before the size of the list was 4. Now the size of the Singly Linked List will be changed and the head of the Singly Linked List will be changed |
|  | Now, the last node of the list is Kiet. | In this function, I will remove the last player from the list. When the application is executed, I will choose 4 numbers to perform and remove the last function. The removeLast() function of the Singly Linked List will be performed. After removing successfully the application displays the player removed and display the size of the list as 2 because, before the size of the list was 3. Now the size of the Singly Linked List will be changed and the last node of the Singly Linked List will be changed. |
|  | Now, the list has 2 players. Duy is the first player in the list and Khoi is the last player in the list. Now, I will get the first player of list is Duy | In this function, I will get the first player from the list. When the application is executed, I will choose 5 numbers to perform and get the first function. The getFirst() function of the Singly Linked List will be performed. After getting successfully the application displays the information of that player. Now the size and the head of the Singly Linked List will keep stable and do not be changed |
|  | Now, the list has 2 players. Duy is the first player in the list and Khoi is the last player in the list. Now I will get the last of list is Khoi | In this function, I will get the last player from the list. When the application is executed, I will choose 6 numbers to perform and get the first function. The getLast() function of the Singly Linked List will be performed. After getting successfully the application displays the information of that player. Now the size and the last of the Singly Linked List will keep stable and do not be changed |
|  |  | In this function, I will check the size of the list. When the application is executed, I will choose 7 numbers to perform this function. The size() function of the Singly Linked List will be performed. Then, the application will display the size of the list. This function will not affect to the Singly Linked List because this function just returns the size of the Singly Linked List. |
|  | If the list has any player    If the list hasn’t player | In this function, I will check whether the list is empty or not. When the application is executed, I will choose 8 numbers to perform this function. The isEmpty() function of the Singly Linked List will be performed. Then, the application will display the message "List is not empty" if the list has the players. But if the list hasn’t players that application will display the message “List is empty”. The Singly Linked List will not affect because this function just returns the Boolean value. |
|  |  |  |
|  |  | In this function, I will show all the players on the list. When the application is executed, I will choose 9 numbers to perform this function. The iter() function of the Singly Linked List will be performed. Then, the application will display all players on the list. The Singly Linked List will not affect because this function just returns the players in the list. |

# IV. Illustrate, with an example, a concrete data structure for a First In First out (FIFO) queue

A queue is described as a linear data structure with open ends and FIFO (First In, First Out) execution of operations. An abstract data structure resembling stacks is the queue. A queue is open on both ends, unlike stacks. It is customary to insert data (an offer) into one end and withdraw it from the other (poll). The data item that was stored first will be accessible first according to the First-In-First-Out principle used by queues. Picture 13 serves as an illustration of a queue (tutorialspoint, 2022)



Figure 13. Example about queue

A real-world example of a queue can be a single-lane one-way road, where the vehicle enters first, and exits first.

There are two famous supported functions of the queue:

* **void offer(E element):** This function is used to add an element to the queue and increase the size of queue
* **E poll():** This function is used to remove an element of the queue and return that element removed, then decrease the size of the queue

Exception of the queue: There is one exception in the queue. When we perform the poll function to remove the element from the queue and the peek function to get the element of queue, but the queue does not have any element in it. The queue will throw an exception to notify the queue is empty.

According to (geeksforgeeks, 2022), the queue has the following advantages and disadvantages

* **Advantages of the queue:**
* Manage efficiently and easily, a large amount of data
* Operations such as insertion and deletion can be performed with ease as it follows the first in first out rule.
* Queues are useful when a particular service is used by multiple consumers.
* Queues are fast in speed for data inter-process communication.
* Queues can be used in the implementation of other data structures.
* **Disadvantages of the queue:**
* Take lots of time for the operations such as insertion and deletion of elements from the middle
* In a classical queue, a new element can only be inserted when the existing elements are deleted from the queue.
* Searching for an element takes O(N) time.
* The maximum size of a queue must be defined prior.

I use Queue ADT to perform a solution for the cinema. When the cinema sells movie tickets and they want to discount the first five tickets so the queue is very suitable for this case because the queue works on the principle of First In First Out (FIFO) and the board of the cinema will know who is the person that buys the tickets in first five people and discount for their tickets. In addition, I choose the queue to use because it is easy to use and with the queue, I can manage the data easily. Besides, the queue is useful and convenient for some cases like the above and has fast data access speed.

First, I generate a random permutation using The Fisher–Yates shuffle function, as shown in Figure 14.

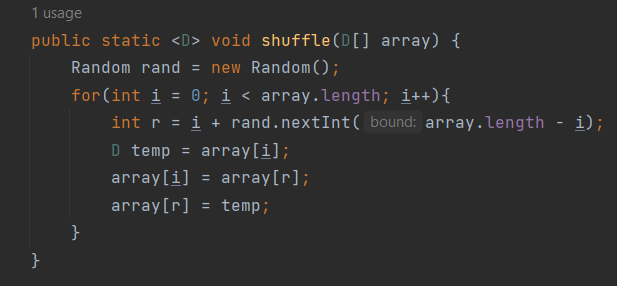


Figure 14. Code random of shuffle method

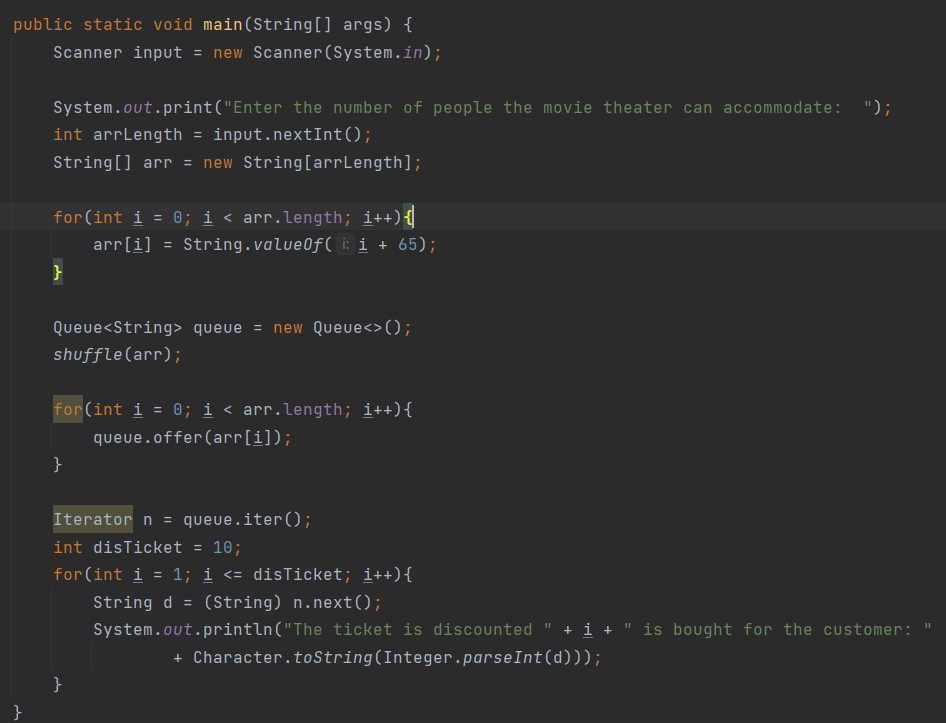


Figure 15. Code of queue about the problem buy the movie ticket with discount cost

In this case, use the offer method of the queue to perform adding the customer to the queue. I will add the customers to an array with the String type. Then, I will transmit this array to the shuffle function to randomize the customers in the array. Next, I use for loop to perform add the customers to the queue by offer() function of the queue. Then, I will use the iter() function of the queue to show the first five customers who buy the movie ticket first.

The list of 5 customer that buy movie ticket first in 20 customers that buy the movie ticket

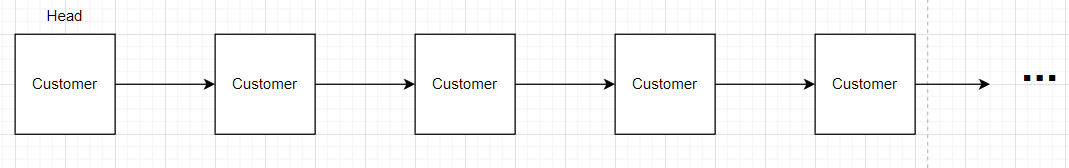


Figure 16. The first five element of queue

Now, I will get the first customer who bought the movie ticket with the discount cost

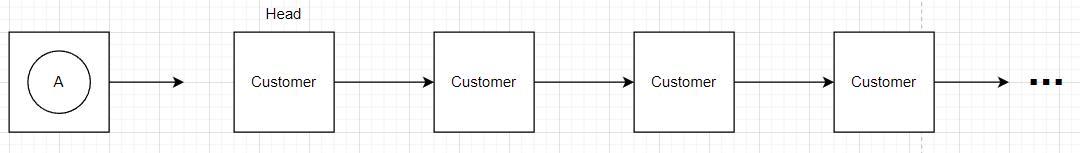


Figure 17. Get the first element of queue

Next, I will get the second customer who bought the movie ticket with the discount cost

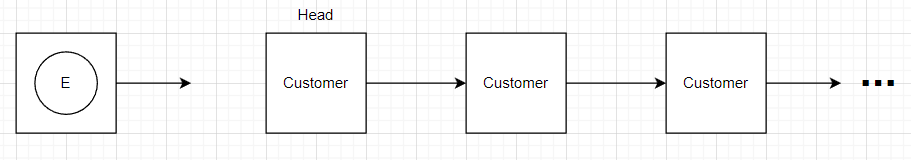


Figure 18. Ge the second element of queue

I will get the third customer who bought the movie ticket with the discount cost

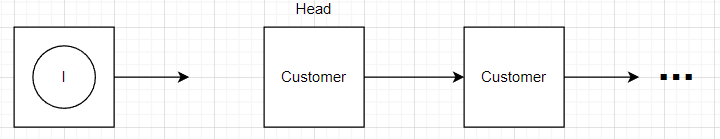


Figure 19. Ge the third element of queue

I will get the fourth customer who bought the movie ticket with the discount cost

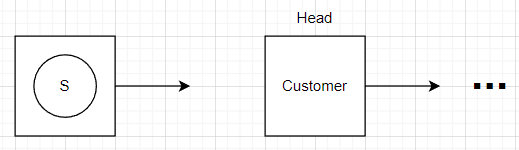


Figure 20. Ge the fourth element of queue

I will get the fifth customer who bought the movie ticket with the discount cot

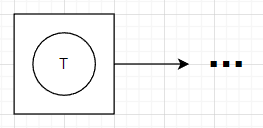


Figure 21. Ge the fifth element of queue

The result of the list customers who bought the movie ticket with discount cost

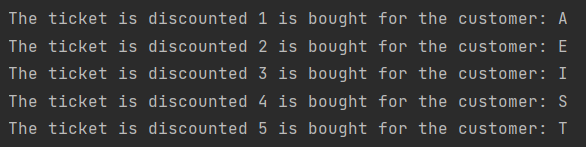


Figure 22. Result console application

# V. Compare the performance of two sorting algorithms

## 1. Sorting algorithms

Sorting is the act of rearranging the provided information into a specific order. This order can be based on the length of the text or on different coding schemes like ASCII or Unicode. It can also be related to numerical values, such as ascending or descending order, alphabetical, such as case sensitive or insensitive. In order to improve the efficiency of the procedure, sorting is a further step that is taken. For instance, organizing a data structure like an array beforehand could speed up a search process. Some algorithms must also work in order to function; for instance, binary search only works with sorted data. To sum up, understand simply sorting algorithms are used to sort the data or elements of array or list them in ascending or descending order (devopedia, 2022)

## 2. Bubble sort

The bubble sort is the most basic sorting technique in some sort algorithms such as Quick sort, Selection sort, and Insert sort. The bubble sort works by continually comparing two adjacent elements and pushing the smaller element to the front and the greater element to the behind. Due to this, each element is compared to its neighboring element, and replacements are made if necessary two adjacent elements (javatpoint, 2021)

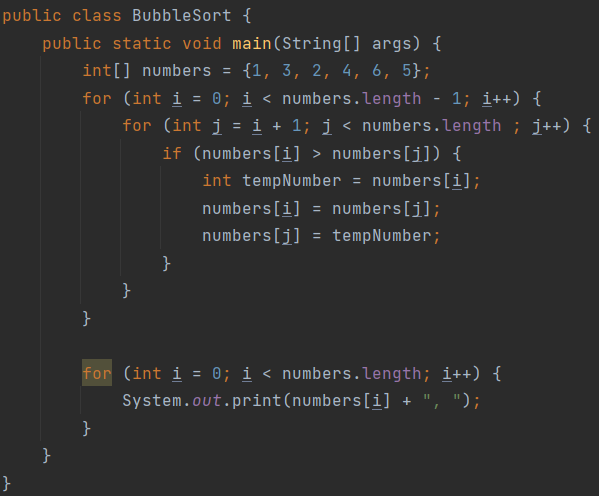


Figure 23. Code bubble sort

* This is an unsorted array

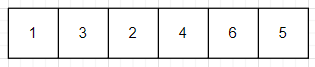


Figure 24. The unsorted array

Now, we will use bubble sort algorithms to sort the elements of the array in increasing

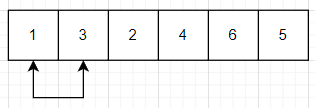


Figure 25. The array is sorting by compare the two numbers 1 and 3

We will compare the number 1 and number 3, but we will keep stable because the number 1 is smaller than the number 3 so the position of them is suitable and we will not sort them

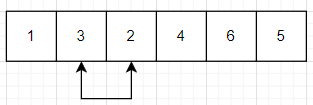


Figure 26. The array is sorting by compare the two numbers 3 and 2

Now, we will switch these two numbers, because the number 3 is greater than the number 2, so we will switch the position of them

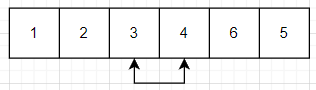


Figure 27. The array is sorting by compare the two numbers 3 and 4

Now, we will compare the number 3 and the number 4, but we will keep stable because the number 3 is smaller than the number 4, so the position of them is suitable and we will not sort them

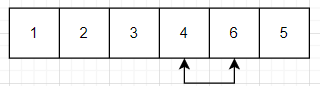


Figure 28. The array is sorting by compare the two numbers 4 and 6

Now, we will compare the number 4 and the number 6, but we will keep stable because the number 4 is smaller than the number 6, so the position of them is suitable and we will not sort them

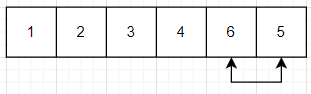


Figure 29. The array is sorting by compare the two numbers 6 and 5

Now, we will compare the number 6 and the number 5. Then, we will switch these two numbers because the number 6 is greater than the number 5, so we will switch the position of them

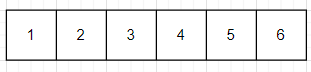


Figure 30. Result of the array after sorting

Now, we sorted the array in increasing the elements in the array

**Performance of bubble sort (Time and Memory)**

According to the (simplilearn, 2022), the performance of the following bubble sort algorithms

The Time Complexity of the Bubble Sort Algorithm: Bubble sort employs two loops: an inner loop and an outer loop. The inner loop performs O(n) comparisons deterministically.

* **Worst case:**

In the worst-case script, the external loop runs O(n) times. As a result, the worst-case time complexity of bubble sort is O( n x n) = O( n x n)( n2).

* **Best case:**

With the array sorted. But if in the case that the bubble sort performs O( n) comparisons. The time complexity of the bubble sort by the best case is O( n).



Figure 31. The calculation of the best case and average case of the bubble sort

The Space Intricacy of the Bubble Sort Calculation: Bubble sort requires just a proper measure of additional room for the variables, i, and size factors. Subsequently, the space intricacy of bubble sort is O. It is an in-place sorting algorithm, which modifies the original array's elements to sort the given array.

## 3. Quick sort

The most efficient sorting algorithm is Quick sort, which completes comparisons in O(nlogn) comparisons. Quick sort also employs the divide and conquer strategy, just like Merge sort (javatpoint, 2021)

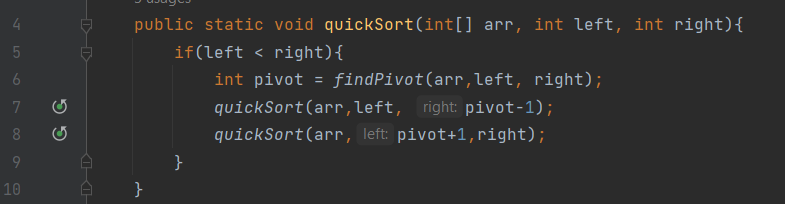


Figure 32. Code of quick sort

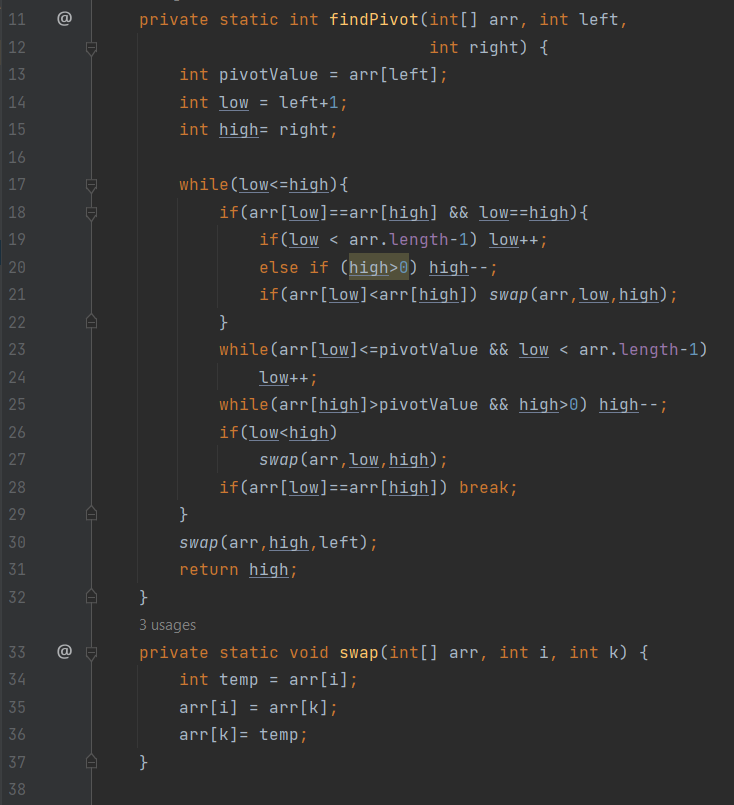


Figure 33. Code of quick sort

Quick sort is the algorithm that is used popularly. For the quick sort, we often use recursion to solve the problem. Firstly, we need to choose a number of arrays to set the pivot. We can set the pivot as the first element or the last element of an array, here I will set the pivot as the first element of the array. Now, I will use the loop at two positions of the array the first position is the left of the array and the second position is the right of the array. Now, I will check, if the value of the left of the array is greater than the pivot, I will swap the value of the right of the array to be smaller than the pivot and continue to like that until the left variable of the array is greater or equal the right variable of the array. When we finish finding the pivot. We will separate the original array into two arrays. In that, the first array will contain the elements that have values that are smaller than the pivot, and the remaining array will contain the elements that have values that are greater than the pivot. Then, we will use recursion to sort the positions in the array by the pivot and continue to separate the array until cannot separate. Each separates array, we will set the new pivot for the array just separated and continue to perform the next steps to sort. Picture 33 is an example of quick sort.

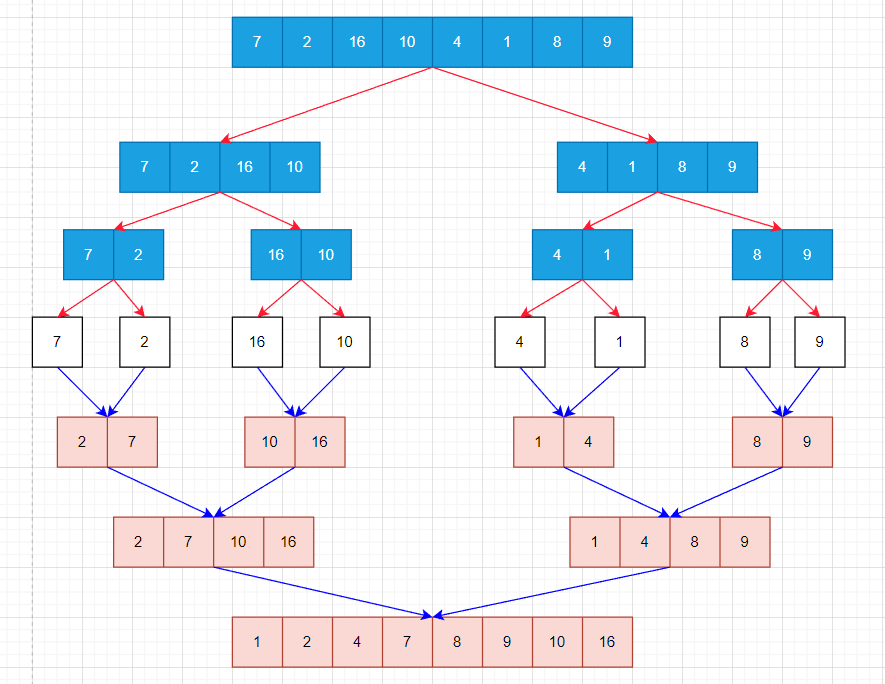


Figure 34. Example about Quick Sort

**Performance of quick sort (Time and Memory)**

According to the (wikipedia, 2022), the performance of the following quick sort algorithms

The Time Complexity of the Quick Sort Algorithm:

* **Worst case:**

In the worst case, when one of the sublists the partitioning algorithm returns a size of n- 1. This might be in some executions if all the values are equal or if the pivot occurs to be the lowest or largest entry in the list. Each recursive call processes a list of size one lower than the former list if this occurs again in every partition. So, before we get to a list of size 1, we can make an n- 1 nested call. As a result, the call tree is basically a direct chain of n- 1 nested calls. In such a script, quicksort takes O( n2) time because the ith call performs O( n2) work to perform the partition.

* **Best case:**

In the best case of the quick sort, every time we partition, the list is resolved into two roughly equal corridors. Consequently, each recursive call processes a list that's half as long. So, before we reach a list of size 1, we can only make log2 n nested calls. This indicates that the call tree's depth is log2n. still, no two calls at the same position of the call tree process the same portion of the original list; as a result, each position of calls requires only O( n) time all at formerly( although each call has some constant outflow, this is incorporated into the O( n) factor since there are only O( n) calls at each position). As a result, the fashion only requires O( n log n) time

* **Average case:**

In the average case, Quicksort averages over all n! permutations of n elements with equal probability in order to sort an array of n unique elements in O(n log n) time in expectation.

# VI. Examine the advantages of encapsulation and information hiding when using an ADT

## 1. What is Encapsulation and Information Hiding

Encapsulation and data hiding are both crucial ideas in object-oriented programming. Encapsulation refers to enclosing data members and method implementation inside a class. The method name can only specify the action that it can carry out on an object of that class when the implementation of all the data members and methods inside a class is encapsulated. Data hiding refers to securing class members from unwanted or illegal access. For example, in this case in part 5 of III. Encapsulation is merging the Node and Element in a class. Information hiding is set the private for the Element (techdifferences, 2022)

## 2. The differences between encapsulation and information hiding

According to (techdifferences, 2022), there are some different between Data Hiding and Encapsulation.

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Basic For Comparison | Data Hiding | Encapsulation |
| 1 | Basic | Along with information hiding complexity, data hiding raises questions regarding data security | Encapsulation involves encasing data to hide a system's complexity |
| 2 | Focus | The goal of Data Hiding is to limit or allow the use of the data that is included inside the capsule. | The goal of encapsulation is to enclose or surround complex data. |
| 3 | Access Specifier | Data that has been hidden is never public and is always private. | The data that is being encapsulated may be public or private. |
| 4 | Process | Data concealing is both a method and a process. | Data concealing includes the sub-process of encapsulation. |

## 3. What do I use to represent an ADT and why

I use ADT to solve the problem about manage players in part 5 of III. ADT is similar to an Interface. When we perform a method in ADT, it just has the name of that method but does not has the body of the method. As a problem about players' management of the club in the problem at part 5 of III. The ADT just perform some method such as adding the player, removing the player, checking the size, checking empty, and printing all players. When implementing ADT, we need a class to implement the operations contained for the function body. Besides, the class to implement the ADT have to perform and comply with all the functions of ADT. For example, when we signed a contract, we have to comply with all that contract requires. As the same ADT, when we use a class to implement the ADT, we have to perform all functions of that ADT.

## 4. When using an ADT, what is the role of Encapsulation and Information Hiding

When I use ADT to solve the problem in part 5 of III, the role of Encapsulation and Information Hiding is as follows. Encapsulation is all methods and attributes of the node that are used and shared in a class. Thanks to this, I will manage the nodes easily. For example, in this problem in part 5 of III, I have to use the getFirst() method to get the data of the first node in the Singly Linked List. In this method, I used the access modifier as private for the nodes. This helps secure the data and avoid the data being accessed by outside. Due to this, when I want to get the element of the node I have to use the getElement() to get. This is the role of Information Hiding in ADT. Besides, the Encapsulation, all of the methods and attributes of the node are shared and used in a class. This is convenient and useful.

# References

devopedia, 2022. *devopedia.org.* [Online]   
Available at: https://devopedia.org/sorting-algorithms  
[Accessed 7 10 2022].

geeksforgeeks, 2022. *geeksforgeeks.org.* [Online]   
Available at: https://www.geeksforgeeks.org/abstract-data-types/  
[Accessed 4 10 2022].

geeksforgeeks, 2022. *geeksforgeeks.org.* [Online]   
Available at: https://www.geeksforgeeks.org/what-is-linked-lista  
[Accessed 5 10 2022].

geeksforgeeks, 2022. *geeksforgeeks.org.* [Online]   
Available at: https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-queue/  
[Accessed 6 10 2022].

Hu, Z., 2022. *zhu45.org.* [Online]   
Available at: https://zhu45.org/posts/2017/Jul/30/understanding-how-function-call-works/  
[Accessed 7 10 2022].

javatpoint, 2021. *javatpoint.com.* [Online]   
Available at: https://www.javatpoint.com/how-to-call-a-method-in-java  
[Accessed 6 10 2022].

javatpoint, 2021. *javatpoint.com.* [Online]   
Available at: https://www.javatpoint.com/sorting-algorithms  
[Accessed 7 10 2022].

liquisearch, 2022. *liquisearch.com.* [Online]   
Available at: https://www.liquisearch.com/abstract\_data\_type/defining\_an\_abstract\_data\_type\_adt/imperative\_abstract\_data\_type\_definitions  
[Accessed 8 10 2022].

simplilearn, 2022. *simplilearn.com.* [Online]   
Available at: https://www.simplilearn.com/tutorials/data-structure-tutorial/bubble-sort-algorithm#what\_is\_a\_bubble\_sort\_algorithm  
[Accessed 8 10 2022].

techdifferences, 2022. *techdifferences.com.* [Online]   
Available at: https://techdifferences.com/difference-between-data-hiding-and-encapsulation.html  
[Accessed 8 10 2022].

techopedia, 2022. *techopedia.com.* [Online]   
Available at: https://www.techopedia.com/definition/9523/stack  
[Accessed 4 10 2022].

tutorialspoint, 2022. *tutorialspoint.com.* [Online]   
Available at: https://www.tutorialspoint.com/data\_structures\_algorithms/dsa\_queue.htm  
[Accessed 6 10 2022].

wikipedia, 2022. *en.wikipedia.org.* [Online]   
Available at: https://en.wikipedia.org/wiki/Quicksort#Formal\_analysis  
[Accessed 8 10 2022].

Yiu, J., 2015. *sciencedirect.com.* [Online]   
Available at: https://www.sciencedirect.com/topics/engineering/stack-memory  
[Accessed 6 10 2022].