Higher Nationals in Computing

Unit 19: Data Structures & Algorithms

ASSIGNMENT 2

Learner's name: TRINH THI DIEU HUYEN

Assessor name: **HO HAI VAN**

Class: GCS0801B.1

ID: GDD18606

Subject code: 1649

Assignment due: Assignment submitted:

ASSIGNMENT 2 BRIEF

Qualification	**BTEC Level 5 HND Diploma in Business**		
Unit number	Unit 19: Data Structures and Algorithms		
Assignment title	Implement and assess specific DSA		
Academic Year			
Unit Tutor	Ho Hai Van		
Issue date		**Submission date**	
IV name and date			

Submission Format:
Format: The submission is in the form of an individual written report. 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 Tutors. The form of submission will be a soft copy in PDF posted on corresponding course of http://cms.greenwich.edu.vn/ Project also needs to be submitted in zip format.

Note: The Assignment *must* be your own work, and not copied by or from another student or from

books etc. 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 know how to reference properly, and that understand the guidelines on plagiarism. *If you do not, you definitely get fail*

Assignment Brief and Guidance:

Scenario: Continued from Assignment 1.

Tasks

For the middleware that is currently developing, one part of the provision interface is how message can be transferred and processed through layers. For transport, normally a buffer of queue messages is implemented and for processing, the systems require a stack of messages.

The team now has to develop these kinds of collections for the system. They should design ADT / algorithms for these 2 structures and implement a demo version with message is a string of maximum 250 characters. The demo should demonstrate some important operations of these structures. Even it's a demo, errors should be handled carefully by exceptions and some tests should be executed to prove the correctness of algorithms / operations.

The team needs to write a report of the implementation of the 2 data structures and how to measure the efficiency of related algorithms. The report should also evaluate the use of ADT in design and development, including the complexity, the trade-off and the benefits.

Learning Outcomes and Assessment Criteria

Pass	Merit	Distinction
LO3 Implement complex data structures and algorithms		**D3** Critically evaluate the complexity of an implemented ADT/algorithm
P4 Implement a complex ADT and algorithm in an executable programming language to solve a well-defined problem. **P5** Implement error handling and report test results.	**M4** Demonstrate how the implementation of an ADT/algorithm solves a well-defined problem	
LO4 Assess the effectiveness of data structures and algorithms		**D4** Evaluate three benefits of using implementation independent data structures
P6 Discuss how asymptotic analysis can be used to assess the effectiveness of an algorithm **P7** Determine two ways in which the efficiency of an algorithm can be measured, illustrating your answer with an example.	**M5** Interpret what a trade-off is when specifying an ADT using an example to support your answer	

Table of Contents

ASSIGNMENT 2 BRIEF 2

Table of Figures 6

Table of Tables 6

ASSIGNMENT 2 ANSWERS 1

LO3 Implement complex data structures and algorithms. 1

P4 Implement a complex ADT and algorithm in an executable programming language to solve a well-defined problem. 1

1. The well-defined problem 1

2. Implement a complex ADT and algorithm in an executable programming language to solve a well-defined problem 1

P5 Implement error handling and report test results. 3

M4 Demonstrate how the implementation of an ADT/algorithm solves a well-defined problem. 5

D3 Critically evaluate the complexity of an implemented ADT/algorithm 6

LO4 Assess the effectiveness of data structures and algorithms 7

P6 Discuss how asymptotic analysis can be used to assess the effectiveness of an algorithm. 8

P7 Determine two ways in which the efficiency of an algorithm can be measured, illustrating your answer with an example. 10

M5 Interpret what a trade-off is when specifying an ADT using an example to support your answer. 12

D4 Evaluate three benefits of using implementation independent data structures 13

REFERENCES 14

Table of Figures

Figure 1: Examples of trees 1
Figure 2: Example of a max-heap (1) 2
Figure 3: Example of a max-heap (2) 2
Figure 4: Implement a min-heap priority queue and use integer for the data structure by Java 3
Figure 5: Implement error handling and report test results by Java (1) 4
Figure 6: Implement error handling and report test results by Java (2) 5
Figure 7: Implement a Task class and priority queue to logging task and level by Java language 6
Figure 8: A full binary tree with N leaves contains 2N - 1 node 6
Figure 9: The time complexity in the case of implementation for the priority queue 9
Figure 10: Implementation Java Code measure the time it takes to queue a new element on an existing 100000000 elements in the queue 11
Figure 11: For example 11

Table of Tables

Table 1: Some the big O most common notation and examples 8

ASSIGNMENT 2 ANSWERS

LO3 Implement complex data structures and algorithms.

P4 Implement a complex ADT and algorithm in an executable programming language to solve a well-defined problem.

1. The well-defined problem

In reality, we have to queue up jobs to do over a period of time and there will be jobs is given first do priority or leaved to do later. So, the problem give is how to associate each job with a different priority and job with higher priority will get to do first. Besides, must store all these jobs in a list and every time add a new a job to this list, it should get updated and put the job with the highest priority on top. In addition, can use list to retrieve this job and update the next job with highest priority. Abstract data type will help us to can do the above.

2. Implement a complex ADT and algorithm in an executable programming language to solve a well-defined problem

The above issue is called a priority queue data structure, it works based on the priority of each item in the list. Item with highest priority will be on the head of the queue which is ready to be retrieved first and then item with the next highest priority. To implement priority queue, need research what is a heap data structure. A heap is a tree-based data structure and specifically a binary tree, such that the nodes are stored based on a specific of order. A tree data structure is like a real-life tree but upside-down where the root is on top and the leaves spread down.

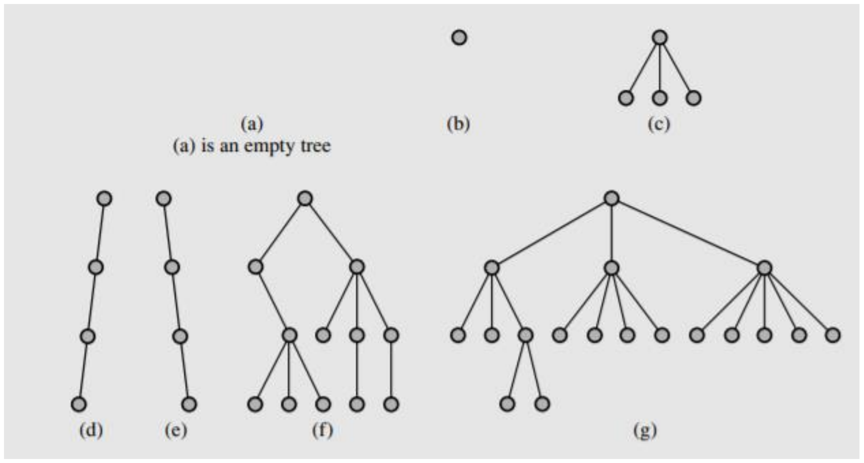


Figure 1: Examples of trees

Usually, there are two types of heap are max-heap and min-heap. Max-heap keeps element with largest value as the root of the tree. Min-heap keeps element with smallest value as the root of the tree. Can use either a min-or-max heap to implement a priority queue. I will use a min-heap to implement the queue, therein smaller values indicate higher priority.

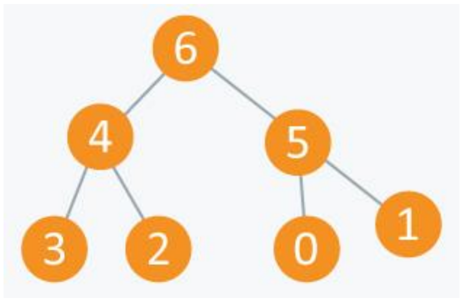


Figure 2: Example of a max-heap (1)

From the above example, can see that at any node, its value is always larger than any of its children and that can help speeding up insertion of new value into the tree. In a binary tree, there are two children nodes of a parent node called the left and right children, can use index in an array, starts at position 1 instead of 0, to represent them. Though not just either smallest or largest value is on the top. As mentioned earlier, they are also stored in a specific order.

- <u>For example</u>: *If a node is at index i, then its left and right children have index 2i and 2i + 1*

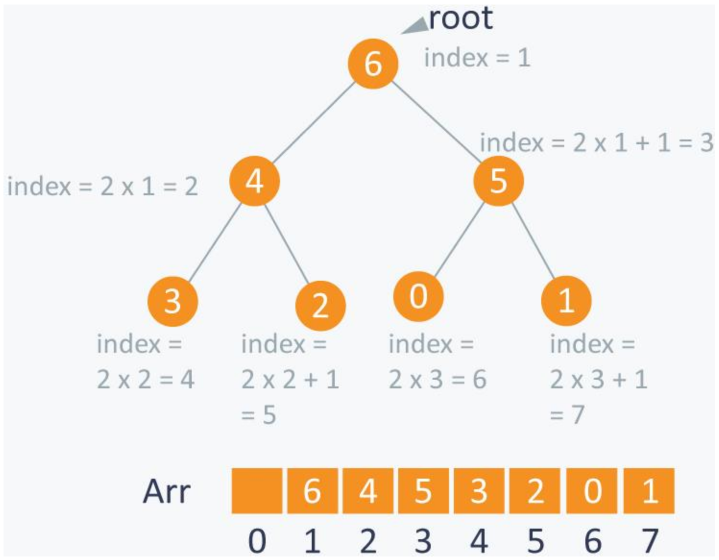
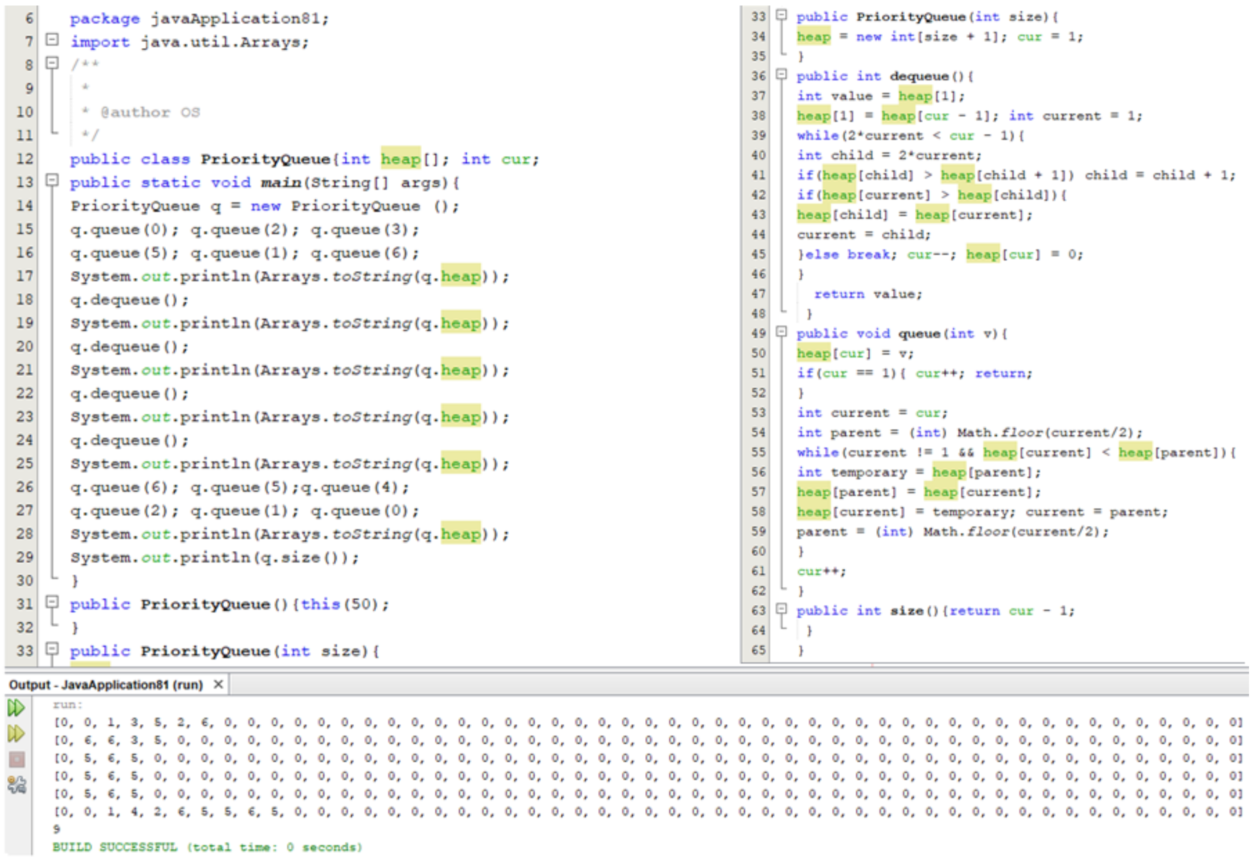


Figure 3: Example of a max-heap (2)

Though we can use dynamic array to implement min-heap but in this case, we will implement a heap using a fixed size array.



```java
package javaApplication81;
import java.util.Arrays;
/**
 *
 * @author OS
 */
public class PriorityQueue{int heap[]; int cur;
public static void main(String[] args){
PriorityQueue q = new PriorityQueue ();
q.queue(0); q.queue(2); q.queue(3);
q.queue(5); q.queue(1); q.queue(6);
System.out.println(Arrays.toString(q.heap));
q.dequeue();
System.out.println(Arrays.toString(q.heap));
q.dequeue();
System.out.println(Arrays.toString(q.heap));
q.dequeue();
System.out.println(Arrays.toString(q.heap));
q.dequeue();
System.out.println(Arrays.toString(q.heap));
q.queue(6); q.queue(5);q.queue(4);
q.queue(2); q.queue(1); q.queue(0);
System.out.println(Arrays.toString(q.heap));
System.out.println(q.size());
}
public PriorityQueue(){this(50);
}
public PriorityQueue(int size){
heap = new int[size + 1]; cur = 1;
}
public int dequeue(){
int value = heap[1];
heap[1] = heap[cur - 1]; int current = 1;
while(2*current < cur - 1){
int child = 2*current;
if(heap[child] > heap[child + 1]) child = child + 1;
if(heap[current] > heap[child]){
heap[child] = heap[current];
current = child;
}else break; cur--; heap[cur] = 0;
}
  return value;
 }
public void queue(int v){
heap[cur] = v;
if(cur == 1){ cur++; return;
}
int current = cur;
int parent = (int) Math.floor(current/2);
while(current != 1 && heap[current] < heap[parent]){
int temporary = heap[parent];
heap[parent] = heap[current];
heap[current] = temporary; current = parent;
parent = (int) Math.floor(current/2);
}
cur++;
}
public int size(){return cur - 1;
 }
}
```

Output - JavaApplication81 (run)

```
run:
[0, 0, 1, 3, 5, 2, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 6, 6, 3, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 5, 6, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 5, 6, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 5, 6, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 4, 2, 6, 5, 5, 6, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
9
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 4: Implement a min-heap priority queue and use integer for the data structure by Java

From the result can see that, the root at position 1 always have the smallest value and each time `dequeue()` is called then the queue updates its new smallest value to the top so it can be later retrieved first.

P5 Implement error handling and report test results.

In Java, if an error is detected during the execution of the program, it will throw an exception. This exception will cause the program to terminate and inform the user that an error has occurred. But if we handle the exception properly, the program can continue without any termination. Java allows this by using try and catch clause. Below is a general format of the `try-catch` statement:

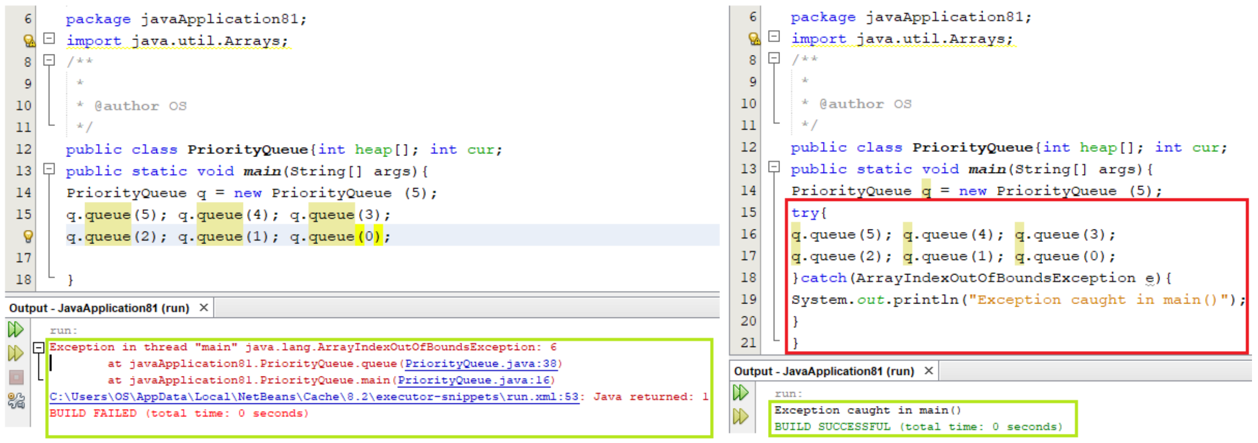
`try` {*do something;*

`}` `catch(`*exception-type exception-name*`)`{*do something;*}

From priority queue implementation has fixed size, can see that it always try to add a new element to the array every time we call `enqueue()`. Means, if the array is already full then adding new element will cause an array out of bound error. In other words, the index that we are trying to access from the array exceed its maximum allowed element. So, we need this error handling by `try-catch` statement. As mentioned, the exception `ArrayIndexOutOfBoundsException` indicates we are accessing an element exceed the array size.

Below is how to handle the exception above:

Before - don't use try-catch **After - use try-catch**



```
package javaApplication81;
import java.util.Arrays;
/**
 *
 * @author OS
 */
public class PriorityQueue{int heap[]; int cur;
public static void main(String[] args){
PriorityQueue q = new PriorityQueue (5);
q.queue(5); q.queue(4); q.queue(3);
q.queue(2); q.queue(1); q.queue(0);

}
```

Output - JavaApplication81 (run)

```
run:
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 6
        at javaApplication81.PriorityQueue.queue(PriorityQueue.java:38)
        at javaApplication81.PriorityQueue.main(PriorityQueue.java:16)
C:\Users\OS\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

```
package javaApplication81;
import java.util.Arrays;
/**
 *
 * @author OS
 */
public class PriorityQueue{int heap[]; int cur;
public static void main(String[] args){
PriorityQueue q = new PriorityQueue (5);
try{
q.queue(5); q.queue(4); q.queue(3);
q.queue(2); q.queue(1); q.queue(0);
}catch(ArrayIndexOutOfBoundsException e){
System.out.println("Exception caught in main()");
}
}
```

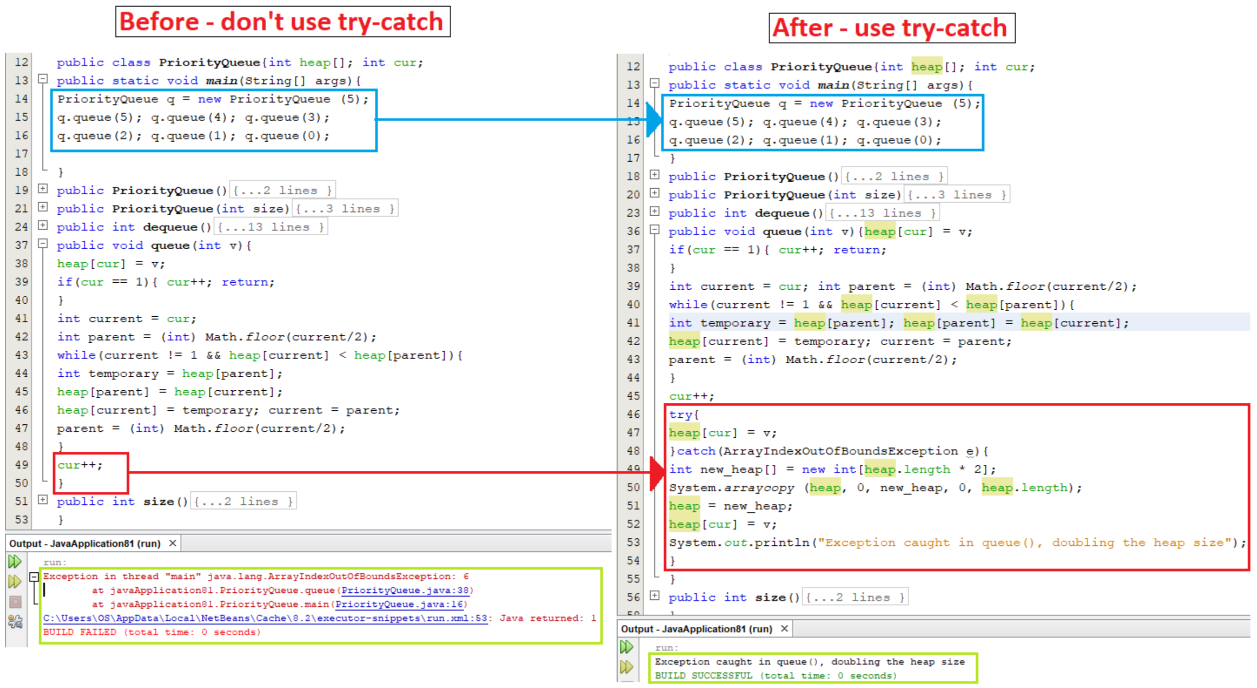
Output - JavaApplication81 (run)

```
run:
Exception caught in main()
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 5: Implement error handling and report test results by Java (1)

It prints out the error to the user when I try to add a sixth element (the exception is raised and it is caught in the `catch` clause). Because we are just informing the user but without changing anything at the queue should handling an error like this does not seem to be very effective. So, it will always report this error. However, still a better approach other that is to update the queue size when an exception is caught to can modify the `queue()` function. Then, `queue()` will first try to attempt adding new element to array, if it fails an exception is raised an caught. This exception will create a new `heap` with doubled size, copy elements from old `heap` and assign the new element to it. Finally, the queue method runs normally because now the size of `heap` will always be doubled when run out of space.

Below is how to implement that by Java language:


Before - don't use try-catch
After - use try-catch
Output - JavaApplication81 (run)

```java
public class PriorityQueue{int heap[]; int cur;
public static void main(String[] args){
PriorityQueue q = new PriorityQueue (5);
q.queue(5); q.queue(4); q.queue(3);
q.queue(2); q.queue(1); q.queue(0);

}
public PriorityQueue(){...2 lines }
public PriorityQueue(int size){...3 lines }
public int dequeue(){...13 lines }
public void queue(int v){
heap[cur] = v;
if(cur == 1){ cur++; return;
}
int current = cur;
int parent = (int) Math.floor(current/2);
while(current != 1 && heap[current] < heap[parent]){
int temporary = heap[parent];
heap[parent] = heap[current];
heap[current] = temporary; current = parent;
parent = (int) Math.floor(current/2);
}
cur++;
}
public int size(){...2 lines }
}
```

```
run:
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 6
        at javaApplication81.PriorityQueue.queue(PriorityQueue.java:38)
        at javaApplication81.PriorityQueue.main(PriorityQueue.java:16)
C:\Users\OS\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

```java
public class PriorityQueue{int heap[]; int cur;
public static void main(String[] args){
PriorityQueue q = new PriorityQueue (5);
q.queue(5); q.queue(4); q.queue(3);
q.queue(2); q.queue(1); q.queue(0);
}
public PriorityQueue(){...2 lines }
public PriorityQueue(int size){...3 lines }
public int dequeue(){...13 lines }
public void queue(int v){heap[cur] = v;
if(cur == 1){ cur++; return;
}
int current = cur; int parent = (int) Math.floor(current/2);
while(current != 1 && heap[current] < heap[parent]){
int temporary = heap[parent]; heap[parent] = heap[current];
heap[current] = temporary; current = parent;
parent = (int) Math.floor(current/2);
}
cur++;
try{
heap[cur] = v;
}catch(ArrayIndexOutOfBoundsException e){
int new_heap[] = new int[heap.length * 2];
System.arraycopy (heap, 0, new_heap, 0, heap.length);
heap = new_heap;
heap[cur] = v;
System.out.println("Exception caught in queue(), doubling the heap size");
}
}
public int size(){...2 lines }
```

```
run:
Exception caught in queue(), doubling the heap size
BUILD SUCCESSFUL (total time: 0 seconds)
```

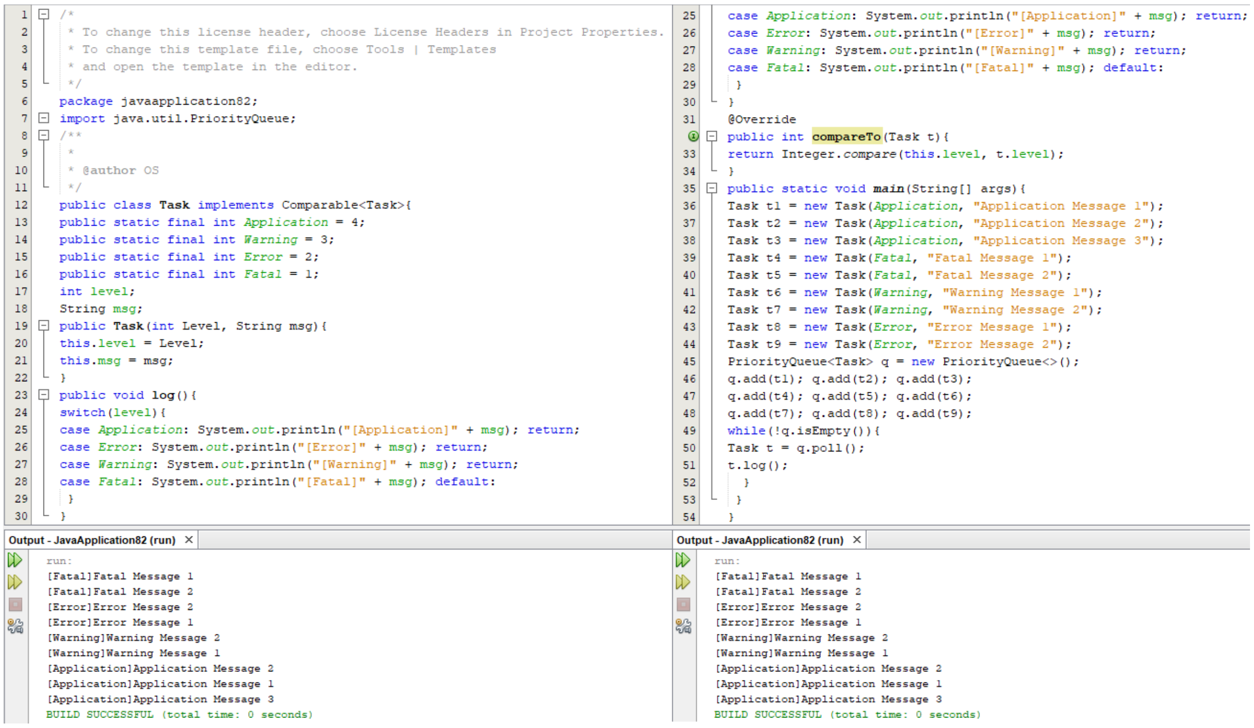
Figure 6: Implement error handling and report test results by Java (2)

M4 Demonstrate how the implementation of an ADT/algorithm solves a well-defined problem.

The well-defined problem is need list that stores tasks with higher priority than another. By implementation of the abstract data type will can demonstrate how it is possible to solve that problem.

Suppose, have a situation fabricate are we logging information to the user in the case of demonstrating the abstract data type then there will be messages of different levels such as Application, Error, Fatal, Warning. From that, messages of fatal level should be informed first due it might indicate that the program will crash and terminate anytime. With error higher than warning then levels error and warning should be informed next due some actions to the program might cause errors or warnings but it also could have been properly handled by the program. In addition, the application level should be informed last to only inform the user successfully taken actions and any further information.

Below is implement a Task class to represent logging task and its level by Java language implementation of priority queue.



```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package javaapplication82;
import java.util.PriorityQueue;
/**
 *
 * @author OS
 */
public class Task implements Comparable<Task>{
public static final int Application = 4;
public static final int Warning = 3;
public static final int Error = 2;
public static final int Fatal = 1;
int level;
String msg;
public Task(int Level, String msg){
this.level = Level;
this.msg = msg;
}
public void log(){
switch(level){
case Application: System.out.println("[Application]" + msg); return;
case Error: System.out.println("[Error]" + msg); return;
case Warning: System.out.println("[Warning]" + msg); return;
case Fatal: System.out.println("[Fatal]" + msg); default:
 }
}
@Override
public int compareTo(Task t){
return Integer.compare(this.level, t.level);
}
public static void main(String[] args){
Task t1 = new Task(Application, "Application Message 1");
Task t2 = new Task(Application, "Application Message 2");
Task t3 = new Task(Application, "Application Message 3");
Task t4 = new Task(Fatal, "Fatal Message 1");
Task t5 = new Task(Fatal, "Fatal Message 2");
Task t6 = new Task(Warning, "Warning Message 1");
Task t7 = new Task(Warning, "Warning Message 2");
Task t8 = new Task(Error, "Error Message 1");
Task t9 = new Task(Error, "Error Message 2");
PriorityQueue<Task> q = new PriorityQueue<>();
q.add(t1); q.add(t2); q.add(t3);
q.add(t4); q.add(t5); q.add(t6);
q.add(t7); q.add(t8); q.add(t9);
while(!q.isEmpty()){
Task t = q.poll();
t.log();
  }
 }
}
```

Output - JavaApplication82 (run)

```
run:
[Fatal]Fatal Message 1
[Fatal]Fatal Message 2
[Error]Error Message 2
[Error]Error Message 1
[Warning]Warning Message 2
[Warning]Warning Message 1
[Application]Application Message 2
[Application]Application Message 1
[Application]Application Message 3
BUILD SUCCESSFUL (total time: 0 seconds)
```

Output - JavaApplication82 (run)

```
run:
[Fatal]Fatal Message 1
[Fatal]Fatal Message 2
[Error]Error Message 2
[Error]Error Message 1
[Warning]Warning Message 2
[Warning]Warning Message 1
[Application]Application Message 2
[Application]Application Message 1
[Application]Application Message 3
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 7: Implement a Task class and priority queue to logging task and level by Java language

From implement above can see fatal messages are guaranteed to be logged first thanks to priority queue. The constructor takes in a level and a message. In addition, it also extends the comparable interface so that it can be added to priority queue for comparison. Other way, the log() function contains a switch clause to output appropriate message according to its level. Summary, priority queue can be applied to different issues due to underlying to priority queue is a heap.

D3 Critically evaluate the complexity of an implemented ADT/algorithm

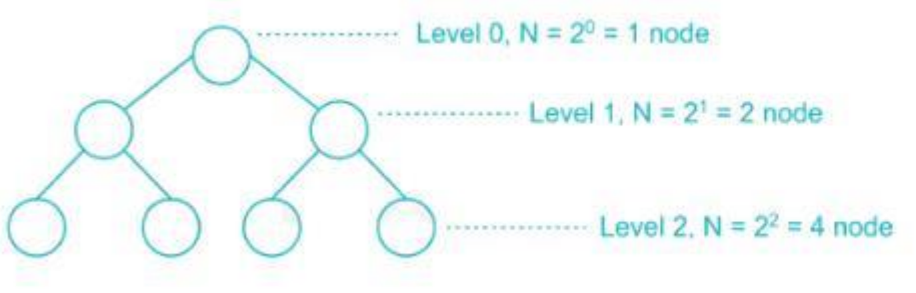

Level 0, N = 2^0 = 1 node
Level 1, N = 2^1 = 2 node
Level 2, N = 2^2 = 4 node

Figure 8: A full binary tree with N leaves contains 2N - 1 node

The time complexity for both method is O(log n) (due to the nature of the tree data structure of the priority queue and the order of elements that the priority queue preserved), the priority queue alone will depend on the implementation of insert and delete elements. In addition, when performing any insertion or deletion of N elements priority queue then the maximum levels that the tree has is where indicates the greatest integer function such that. Follow Figure 8, there are 4 leaf nodes in the above tree. So, total number of nodes are 2 * 4 - 1 = 8 - 1 = 7 elements so the maximum levels that this tree has is three levels due $\log_2 7 + 1 = 3$.

We will comeback Priority Queue program to prove this. When the implemented function `dequeue()` is called then it copies the root element after that puts the last element as the root and before the while loop will the current index is assigned as 1. Next, at each iteration of the while loop then will check whether the child index is still inbound of the array, and it checks whether the parent is larger than its child, this is which is obtained by either "`2*current`" or "`2*current+1`" later that will swap if it is larger and the current index is assigned as either "`2*current`" or "`2*current+1`". Next, the while loop will terminate when either the child of the current index exceeds array bound or the parent is not larger than any of its children. Finally, return the previous root value but the number of times it can travel again is [log n + 1] so the time complexity is O(log n).

When given enough array sized then can also offer a similar approach for `enqueue()` method. First, it adds the new element to the end of the heap and after that current index is assigned as the index that the new element is in with the parent of this current index is calculated by using the greatest integer function `parent=[current/2]` and if the current index is smaller than its parent then the two is swapped. Next, while loop terminates when either current index is the root or the current index is larger than its parent in the case that the current index is the root. Anew, the maximum number of times it can travel is [log n] + 1, so the time complexity O(log n). Although in the case that not having enough size when adding new element then a new array of doubled size is created and copied the original array into it so it takes O(n) time to copy so the time complexity is O(n+logn). Besides, the space complexity for both functions is O(1) so it does not requires any extra space for storing a lot of elements except for when copying in queue() which has O(n) space complexity.

LO4 Assess the effectiveness of data structures and algorithms

P6 Discuss how asymptotic analysis can be used to assess the effectiveness of an algorithm.

When analyze algorithms, we should employ mathematical techniques to that analyze algorithms independently of specific implementations, computers, or data. To analyze algorithms, we have two steps. First, start to count the number of significant operations in a particular solution to assess its efficiency. Later, express the efficiency of algorithms using growth functions. Implementations data comes with running-time result will various for with different computers. So, we can use the asymptotic analysis to assess the effectiveness of the implemented algorithm. In addition, the most inefficient algorithm running on the Cray computer (supercomputer) can perform much better than the most efficient algorithm on a PC due always system-dependent. Thus, it is required to have a way to approximate the running-time of the algorithm independent of those three elements and system. That is where asymptotic analysis can be used. It provides an upper-bound for algorithm that is no matter where it is used. Besides, it can be guaranteed to give the same effect everywhere.

Table 1: Some the big O most common notation and examples

Big O Notation	Name	For examples
O(1)	Constant	Odd or Even number and Look-up table (on average)
O(log n)	Logarithmic	Finding element on sorted array with binary search
O(n)	Linear	Find max element in unsorted array and Duplicate elements in array with Hash Map
O(n log n)	Linearithmic	Sorting elements in array with merge sort
$O(n^2)$	Quadratic	Duplicate elements in array and Sorting with bubble sort
$O(n^3)$	Cubic	Three variables equation solver
$O(2^n)$	Exponential	Find all subsets
O(n!)	Factorial	Find all permutations of a given set/string

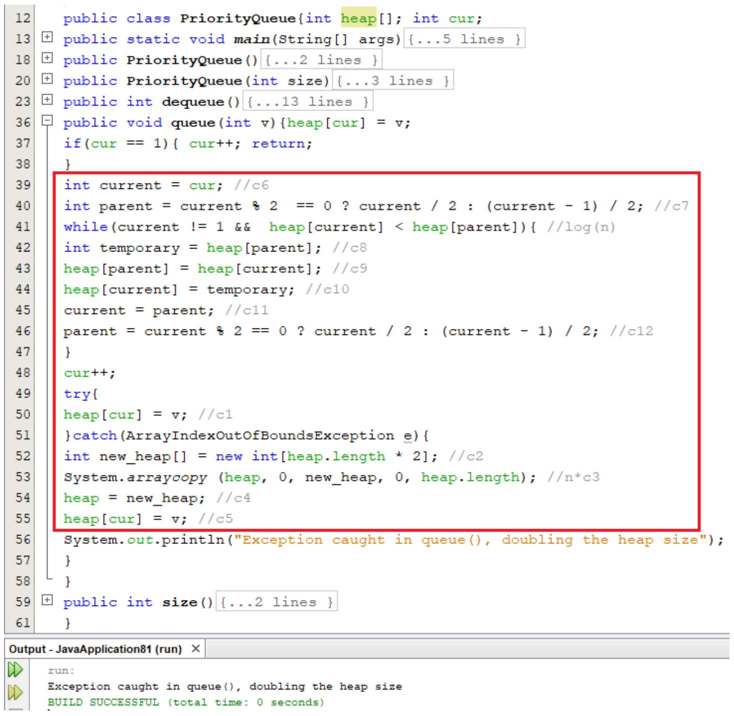
The reason it is called an upper-bound because this is an approximation to the real truth value of the running-time and always larger than the actual running-time given large enough data. In other words, when the data grows large enough, the constant factors in the truth running time function becomes insignificant, therefore, the asymptotic analysis only takes account in for the term that grows significantly with the data. In asymptotic analysis, the running-time is indicated using the big-O notation. Need sum the individual running time complexity of each statement for this algorithm to calculate the time complexity.

In the case that the exception does not occur then when summing all the time complexity can deduced that: $c_1 + c_6 + c_7 + \log_2 n(c_8 + c_9 + c_{10} + c_{11} + c_{12})$

$$= a + b\log_2 n$$

$$= O(\log_2 n)$$

Means, the algorithm running-time is O(log n) when the exception does not occur, we get O(n log n) when it does. The same approach can also be used for `dequeue()` and it also bring O(log n). The cause that the “while loop” loops for log n times is at each iteration the algorithm divides the current index of `current` by 2 but the number of times it can divides before reaching 1 is log n (the number of times). Thus, we can insure that as the number of n grows large then the effect is will defined by the above time complexity.



```
public class PriorityQueue{int heap[]; int cur;
public static void main(String[] args){...5 lines }
public PriorityQueue(){...2 lines }
public PriorityQueue(int size){...3 lines }
public int dequeue(){...13 lines }
public void queue(int v){heap[cur] = v;
if(cur == 1){ cur++; return;
}
int current = cur; //c6
int parent = current % 2  == 0 ? current / 2 : (current - 1) / 2; //c7
while(current != 1 &&  heap[current] < heap[parent]){ //log(n)
int temporary = heap[parent]; //c8
heap[parent] = heap[current]; //c9
heap[current] = temporary; //c10
current = parent; //c11
parent = current % 2 == 0 ? current / 2 : (current - 1) / 2; //c12
}
cur++;
try{
heap[cur] = v; //c1
}catch(ArrayIndexOutOfBoundsException e){
int new_heap[] = new int[heap.length * 2]; //c2
System.arraycopy (heap, 0, new_heap, 0, heap.length); //n*c3
heap = new_heap; //c4
heap[cur] = v; //c5
System.out.println("Exception caught in queue(), doubling the heap size");
}
}
public int size(){...2 lines }
}
```

Output - JavaApplication81 (run)

```
run:
Exception caught in queue(), doubling the heap size
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 9: The time complexity in the case of implementation for the priority queue

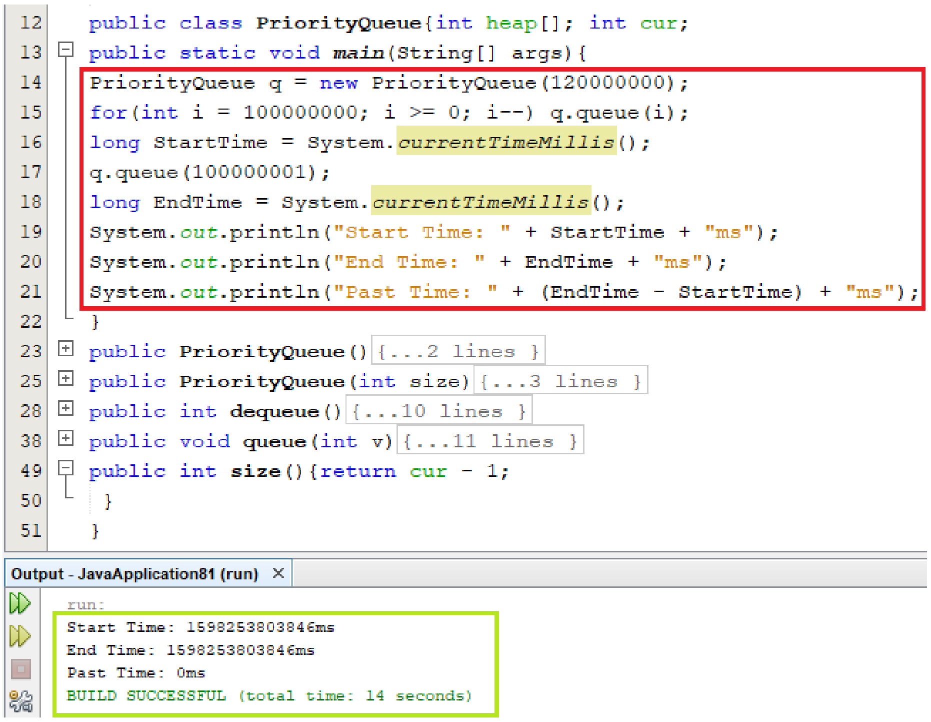
P7 Determine two ways in which the efficiency of an algorithm can be measured, illustrating your answer with an example.

Have two ways to evaluate the effectiveness of the algorithm that are Time and Space. About time, use time complexity analysis to estimate run time as an algorithm of the size of input data and result is usually denoted by the O symbol. This is convenient for comparing algorithms when large amounts of data will be processed. Usually there will be 3 cases that are the best time, average time and worst time to be able to evaluate the performance of an algorithm. In fact, people often use average time to balance time and performance. About space, it relates to the use of memory resources while the algorithm is being implemented and have up to three aspects of memory usage to consider. Firstly, the amount of memory required to store the code for the algorithm. Secondly, memory capacity to store data input. Thirdly, the amount of memory required for data at the output. From this can be see that, the relationship between time and capacity is opposite. The more time an algorithm is executed, the lower the memory capacity will be, and the less time the execution time will be, the higher the memory capacity will be.

Like said above, can measure an algorithm by either space or time. Nowadays, the space complexity issues has already been solved due we can just buy more storage to store more data at a relatively. Opposite, it is not always the case for time complexity as there is a limit for how fast a CPU or GPU can perform. So, this is why people usually focus more on the time performance of an algorithm than space complexity.

- For example: *If an algorithm perform writing onto a fast disk but processing its data truly slow, it would be meaningless for a fast disk due it needs to wait for the algorithm to process. It is sure that an efficient algorithm can lead to a better user experience for it does not make users wait long.*

As mentioned before, we can use the big-O notation to determine an upper-bound (measuring the time complexity) on how fast an algorithm can run on a large number of data. In addition, still way other is see how much time it really takes by using a stopwatch on an algorithm. Based on the examples in previous P sections, I will continue implement code on priority queue's `queue()` to can measure the time it takes to queue a new element on an existing 100000000 elements in the queue.



```
public class PriorityQueue{int heap[]; int cur;
public static void main(String[] args){
PriorityQueue q = new PriorityQueue(120000000);
for(int i = 100000000; i >= 0; i--) q.queue(i);
long StartTime = System.currentTimeMillis();
q.queue(100000001);
long EndTime = System.currentTimeMillis();
System.out.println("Start Time: " + StartTime + "ms");
System.out.println("End Time: " + EndTime + "ms");
System.out.println("Past Time: " + (EndTime - StartTime) + "ms");
}
public PriorityQueue(){...2 lines }
public PriorityQueue(int size){...3 lines }
public int dequeue(){...10 lines }
public void queue(int v){...11 lines }
public int size(){return cur - 1;
 }
}
```

Output - JavaApplication81 (run)

```
run:
Start Time: 1598253803846ms
End Time: 1598253803846ms
Past Time: 0ms
BUILD SUCCESSFUL (total time: 14 seconds)
```

Figure 10: Implementation Java Code measure the time it takes to queue a new element on an existing 100000000 elements in the queue

Although, the new element is the maximum element in the heap. Means, it need to travel to the last level in the tree but the time it takes is basically instant due O(log n) for 100000000 elements is only 27 levels maximum in the tree. Actually, this number is still very small so the insertion will be faster, but it is not always the case to be able to develop an O(log n) time algorithm. There are often cases where the polynomial time algorithm is considered as fast as the problem P and NP.

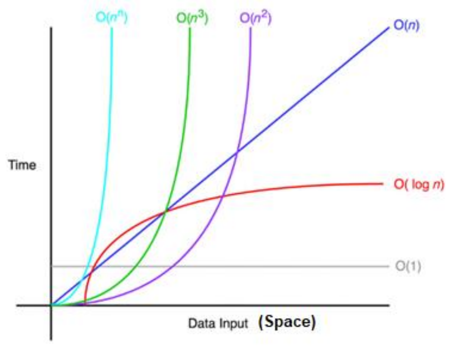


Figure 11: For example

- For example: Based on the chart above we have, n*o comparison-based sorting algorithms that are faster than O(n log n). However, it still considered fast because it has polynomial time of degree one. To can give a better understanding when developing an algorithm then need having an insight on how others time complexity will perform. In fact, often developing algorithms with time complexity O(n^2), O(n^3), O(n^n) because they can scale really fast as the data grows. However, in some cases when the input size is small then these algorithms can have efficient the same. Priority queue is an efficient abstract data type due both insertion and deletion of element runs in logarithmic time. It can be clearly seen that in the chart above then logarithmic time does not scale quickly as others polynomial time complexity.*

M5 Interpret what a trade-off is when specifying an ADT using an example to support your answer.

There are some trade-offs we have to make due to the way the abstract data type is defined when specifying it. First, the abstract data type is not always obvious enough for the programmer to understand, and this may result in the implementation wrongly intended before. Second, when designing an abstract data type, we can only define its behavior and not have implementation specific information. Therefore, when trying to implement that abstract data type for a particular language can cause some confusion for some programmers since it is based on how those programmers interpret or specify describe the abstract data type. On the other hand, abstract data types are often implemented to accommodate more of their defined functionality in the specifications. In addition, depending on which language the abstract data types implement there may also have more than one function over another implementation. Hence, it can cause possible instances of redundant functions.

- For example: *The `pop()` function in the Stack abstract data type specification removed the most recently added element. It doesn't say anything but instead is returns that recently added element. However, still have implementations that both removes and returns the value and extension function that allows to see the top value is also added to avoid removal of the element.*

In a nutshell, the trade-off when specifying an abstract data type is that there may be some loss of information reading about the abstract data type specification but have general freedom of implementation since it is not specific implementation language. Through here can see although it is necessary to trade some benefits, it can help users to use better due it is possible to expand the functionality to support the programmer when just need one type abstract data can also act as a building block for abstract data type others.

D4 Evaluate three benefits of using implementation independent data structures

Three benefits of using implementation independent data structures is Representation Independence, Modularity and Interchangeability of Sections. First, Representation Independence then at most of the program becomes independent of the abstract data type's representation, hence that implementation can be improved without breaking the entire program. Next, Modularity, different with representation independence, here the different parts of a program become less dependent on other parts and on how those other parts are implemented. Finally, Interchangeability of Sections is different implementations of an abstract data type may have different performance characteristics. With abstract data types, it becomes easier for each part of a program to use an implementation of its data types that will be more efficient for that particular part of the program.

- For example: *Java's standard libraries supply several different implementations of its `Map` data type. The `TreeMap` implementation might be more efficient when a total ordering on the keys can be computed quickly but a good hash value is hard to compute efficiently. The `HashMap` implementation might be more efficient when hash values can be computed quickly and there is no obvious ordering on keys. The part of a program that creates a `Map` can decide which implementation to use. The parts of a program that deal with a created `Map` don't have to know how it was implemented; once created, it's just a `Map`. If it weren't for abstract data types, every part of the program that uses a `Map` would have to be written twice, with one version to deal with `TreeMap` implementations and another version to deal with `HashMap` implementations.*

These advantages become more important in larger programs, so they often go under-appreciated by programmers with limited experience.

REFERENCES

[1] Adam Drozek, 2010. *Data Structures And Algorithms In Java*. 2nd ed. [ebook] Available at: <https://pdfs.semanticscholar.org/03fb/c6e2cd9ef11dfe000696849786b372fdd777.pdf?_ga=2.95085468.1647234384.1596762023-1381977231.1596762023> [Accessed 19 August 2020].

[2] Prateek Garg, 2020. *Heaps And Priority Queues - Prateek Garg*. [online] HackerEarth. Available at: <https://www.hackerearth.com/fr/practice/notes/heaps-and-priority-queues/> [Accessed 19 August 2020].

[3] Adrian Mejia, 2019. *8 Time Complexities That Every Programmer Should Know*. [online] Adrian Mejia Blog. Available at: <https://adrianmejia.com/most-popular-algorithms-time-complexity-every-programmer-should-know-free-online-tutorial-course/> [Accessed 19 August 2020].

[4] Bonvic Bundi, 2019. *Understanding Big-O Notation With Javascript - DEV*. [online] Dev.to. Available at: <https://dev.to/b0nbon1/understanding-big-o-notation-with-javascript-25mc> [Accessed 19 August 2020].

[5] GeeksforGeeks. 2016. *Lower Bound For Comparison Based Sorting Algorithms - Geeksforgeeks*. [online] Available at: <https://www.geeksforgeeks.org/lower-bound-on-comparison-based-sorting-algorithms/> [Accessed 19 August 2020].

[6] Faihycape, 2020. *What A Trade-Off Is When Specifying An Adt - Faihycape'S Blog*. [online] faihycape's blog. Available at: <https://klepiswora.hateblo.jp/entry/2020/06/20/060620_1> [Accessed 19 August 2020].

[7] Quora.com. 2020. How Many Nodes Does A Full Binary Tree With N Leaves Contain? - Quora. [online] Available at: <https://www.quora.com/How-many-nodes-does-a-full-binary-tree-with-N-leaves-contain> [Accessed 24 August 2020].

[8] William Clinger, 2020. *Advantages Of Abstract Data Types*. [online] Course.ccs.neu.edu. Available at: <https://course.ccs.neu.edu/cs5010f17/InterfacesClasses2/advantagesADT1.html?> [Accessed 24 August 2020].