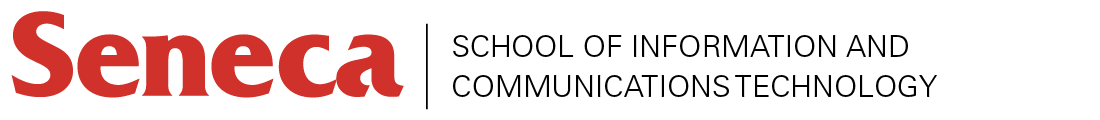
**Test 1 **

|  |  |  |  |
| --- | --- | --- | --- |
| TERM | COURSE NAME | COURSE CODE | VERSION |
| Fall | Data Structures and Algorithms | BTP500 | A |

|  |  |
| --- | --- |
| Name | Khai Phan Thanh |
| Student Number | 100901164 |
| Section | NBB |

DATE: Friday, October 23

TIME ALLOWED: 1 hour 45 min

QUESTIONS:

PART A Short/Long Answers 35 MARKS

PART B Programming 20 MARKS

TOTAL MARKS 55 MARKS

PROFESSOR(S): Catherine Leung, Miguel Watler

**SPECIAL INSTRUCTIONS:**

1. Feel free to use all resource material available to you, including sample code from the lectures, labs, assignments, and internet resources.
2. Answer all questions in the spaces provided on the test paper

This test includes a *cover page*, plus 10 pages of *questions*.

|  |
| --- |
| SENECA’S ACADEMIC HONESTY POLICY |
| As a Seneca student, you must conduct yourself in an honest and trustworthy manner in all aspects of your academic career. A dishonest attempt to obtain an academic advantage is considered an offense, and will not be tolerated by the College. |

Question 1 [4 marks]:

What are sentinel nodes? Where are they (positionally) within an array? What is their purpose?

* **Sentinel Node are empty node located at the front and back of the array. These nodes always exist in the array from the time array is initialize (create) until it ends. These Nodes do not hold data, and their purpose is using to eliminate cases (mostly corner case) when adding or removing data from the array.**

Question 2 [4 marks]:

A singly linked list is a linked list that only has a pointer going forward in each node (ie just next, no prev). As a result, the run time of the pop\_back() function is O(n) instead of O(1). The runtimes of push\_front/push\_back and pop\_front are still O(1) (assuming that there is a pointer to both the first and last node in linked list). If you were using a singly linked list to implement a **Queue**, is it better to use the push\_front() or push\_back() to implement the **enqueue()** function or does it even matter? Explain your answer in 2 or 3 sentences. Be sure to include in your answer what a **Queue** is and why your choice is good.

* **Queue is a special type of array that follow in the order of: First-in, First-out, (like a car going through the tunnel, as Miguel said). And because of its specialty, either push\_front() or push\_back() will work. Once we have determined which end the data go, the dequeue() will follow the end-side to remove the data correctly.**

Question 3 [4 marks]:

At an interview, an interviewer asks you the following question:

"The build time for our application is 1 minute. Last month, when we ran our tests, our program passed every test. Today, when we run our tests, our program fails our tests. Between last month’s testing and today testing there have been 100 commits of the code and one of these commits introduced an error, but we don’t know which one. What is the quickest way of finding which commit it was?"

a) Explain how you would go about figuring out which version had the bug in it, assuming the commits have been numbered 1 to 100.

* **The best way to solve this is break it down into small chunks, preferably 2 equal chunks and test for the data. If there are 100 commits, the order of testing would be:**

**100/2 = 50**

**50/2 = 25**

**25/2 = 12.5**

**12.5/2 = 6.25**

**6.25/2 = 3.125**

**3.125/2 = 1.5625**

* **In the above tree, simplest explanation will be, in the first 100 commit, run them all at the same time, find the tree that contain the error and only work on that tree, from there, we can eliminate the other half which run successfully. Same methodology applies when it gets into single commit testing (last stage).**

b) What is the maximum amount of time (number of minutes) it would take for you to figure out which commit has the bug?

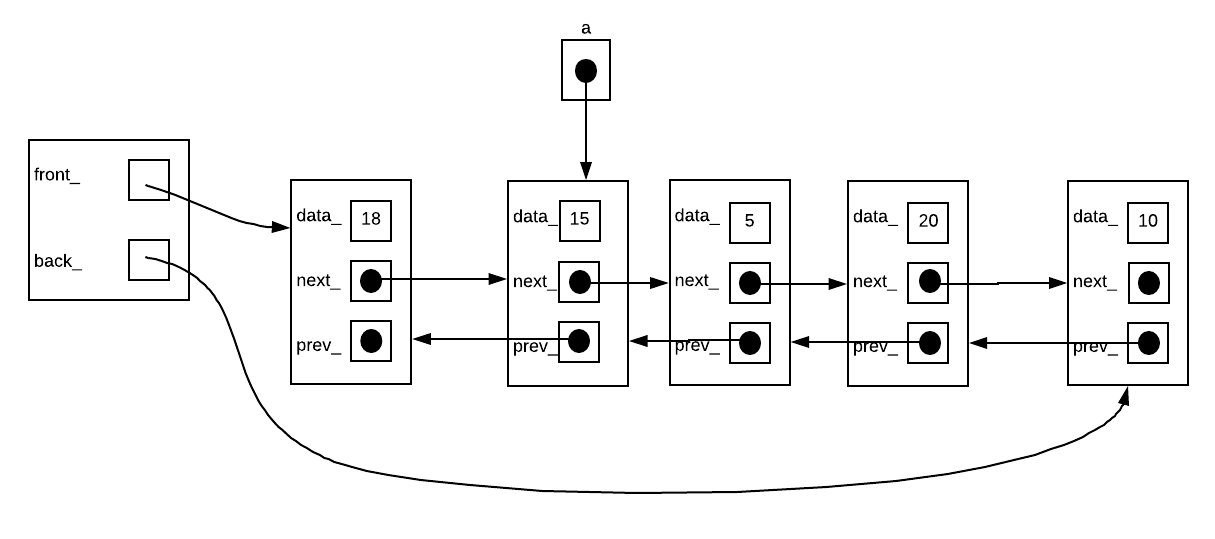
* **There are a total of 6 levels, assume each level takes 1 minute to build the program and at every stage, we will run all possible failure commit. There are a total of 6 minutes at maximum time to determine a failure case, note that we do not build each commit one by one, we do all at the same time.**

Side notes:

* **Build time** is the time it takes to compile the program. In order to test a version, you must build it. Thus each version you test will take 30 min to determine if that particular version is passes testing
* **Commits** refer to a version. 100 commits mean 100 different versions. This is like a commit you make to your github repo.

Question 4 [5 marks]:

The drawing below shows the current state of a linked list.



a) Draw what the list would look like if you were to insert the value 6 into the list ***after*** the node pointed to by ***a***

**Data\_ 15 Data\_ 6 Data\_ 5**

**Next\_ ------------------------🡪 Next\_ --------------------🡪 Next\_ \*\*point to Data\_ 20\*\***

**Prev\_ \*\*point to Data\_ 18\*\* Prev\_ \*\*point to Data\_ 15\*\* Prev\_ \*\*point to Data\_ 6\*\***

b) Does the following blurb of code correctly insert a node with 6 ***after*** the node pointed to by ***a***? If not, rewrite the code blurb so that it works.

//assume constructor is: //Node(int data, Node\* next, Node\* prev)

**//add a new node where next\_ is equal to current a->next\_ and prev\_ is equal to a**

Node\* nn=new Node(6, a->next\_, a);

**//relocate position of a->next\_**

a->next\_=nn;

**//relocate the position of a->next\_->prev\_ to new added node**

a->next\_->prev\_=nn;

* **This blurb of code is correctly added the new node after position of a**

Question 5 [7 marks]:

Below is a table with some key values and their hash indexes. Using this information, perform the given operation in each of the following ***linear probing hash tables*** below

|  |  |
| --- | --- |
| Key | Hash Index |
| apple | 5 |
| orange | 6 |
| banana | 6 |
| grape | 1 |
| pear | 9 |
| kiwi | 8 |
| lichee | 7 |
| mango | 0 |
| cherry | 8 |
| lemon | 5 |
| lime | 2 |
|  |  |

* **Assume the hash% is 10**

a) Insert the following keys into the table in the order given (2 marks)

orange, banana, kiwi, cherry, mango, pear, grape

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Mango** | **Pear** | **Grape** |  |  |  | **Orange** | **Banana** | **Kiwi** | **Cherry** |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

b) remove ***apple*** from the following table. Write out what the new table looks like once you finish the remove operation in the empty table below (2 mark)

**🡪 Assume the order for this one begin at 5**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | apple | lemon | lichee | cherry | banana |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | **Lemon** | **Banana** | **Lichee** | **Cherry** |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

c) remove ***lemon*** from the following table. Write out what the new table looks like once you finish the remove operation in the empty table below (3 marks)

**🡪 Assume the order for this one begin at 0**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| mango | banana | lime |  |  | apple | lemon | orange | cherry | pear |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Mango** |  | **Lime** |  |  | **Apple** | **Banana** | **Orange** | **Cherry** | **Pear** |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Question 6 [6 marks]:

Given the following code, perform an analysis of ***function3()*** where n is the size of the array. To get full marks you must perform a full analysis. Simply saying something like "function3() has a run time of O(n)" will only get you 1 mark. It's the steps that matter. Ensure that the analysis is clear and detailed.

void function1(int& a, int& b);

void function2(int array[],int size);

void function3(int array[],int size);

void function1(int& a, int& b){ O(n) = 5

int tmp = a; //2

a = b; //1

b = tmp; //2

}

void function2(int array[],int size){ O(n)

for(int i = 0;i<size-1;i++){ //2n-1+1

if(array[i] > array[i+1]){ //4n-1

function1(array[i],array[i+1]); //

}

}

}

void function3(int array[],int size){ O(n) =

for(int i=0;i<size;i++){ //3n+1

function2(array,size); //

}

}

Question 7 [5 marks]:

Below is an array. Using words and diagrams (not code), explain how to sort the following array using the **merge sort** algorithm

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 8 | 3 | 7 | 1 | 4 | 6 | 2 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

A close up of text on a whiteboard

Description automatically generated

* **In the above picture, we simply take the big array and divide it into smaller size array, preferably 2 equal size. Then we will continue the same process until each number or value is on its own. After that we will merge them back as how we divide it (ex. If 5-8 is divided into 5 and 8, they are now merging back). While merging, they will sort in order from smallest to greatest, left to right.**

Part B - Programming: Please put all answers in the space provided.

Question 8 [6 marks]:

Write the following function **recursively** without using the multiplication (\*) mathematical operator. Note, a **non-recursive solution** to the following problem will result in a **grade of 0**, even if it is correct.

**unsigned int multiply (unsigned int a, unsigned int b);**

This function returns a\*b. However, you can't use the multiplication (\*) operator your solution

unsigned int multiply (unsigned int a, unsigned int b) {

//Enter code here

unsigned int total = 0;

for (int i = 0; i < b; i++) {

total += a;

multiply(total, i);

}

return total;

}

Question 9 [14 marks]:

In this coding question for linked lists, you may assume that the list uses sentinels or does not use sentinels. The choice is yours. But you must pick**: If you do not pick, I will mark your code assuming the linked list does NOT using sentinels**

**Will your code include the use of sentinels (circle one): YES**

Suppose you were given the following declaration for a node and linked list. **Every function listed below is a function you can call without implementing**. Any function not listed below must be coded by you.

class DList{

Node\* front\_;

Node\* back\_;

struct Node{

int data\_;

Node\* next\_;

Node\* prev\_;

Node(int data = 0, Node\* nx=nullptr, Node\* pr=nullptr){...}

};

public:

class const\_iterator{

Node\* curr\_;

friend class DList;

const\_iterator(Node\* p){...}

public:

const\_iterator(){...}

const\_iterator operator++(){...}

const\_iterator operator++(int){...}

const T& operator\*() const {...}

bool operator==(const\_iterator){...}

bool operator!=(const\_iterator){...}

};

class iterator:public const\_iterator{

...

iterator(Node\* p){...}

friend class DList;

public:

iterator(){...}

iterator operator++(){...}

iterator operator++(int){...}

T& operator\* {...}

};

DList(){...}

const\_iterator begin()const{...}

const\_iterator end()const{...}

iterator begin(){...}

iterator end(){...}

};

Question is on next page.

a) Write the following member function: (6 marks)

void DList::moveToBack(iterator it);

This function will move the node referred to by the iterator ***it*** to the back of the list. The ordering of all other nodes in list must not be altered.

template <typename T>

void Sentinel<T>::moveToBack(iterator it) {

//Enter code here for the sentinel version

std::cout << "Move to Back code" << std::endl;

std::cout << "value to move is: " << it.curr\_->data\_ << std::endl;

it.curr\_->prev\_->next\_ = it.curr\_->next\_;

it.curr\_->next\_->prev\_ = it.curr\_->prev\_;

it.curr\_->next\_ = back\_;

it.curr\_->prev\_ = back\_->prev\_;

back\_->prev\_ = it.curr\_;

back\_->prev\_->next\_ = it.curr\_;

}

b) Write the following member function: (8 marks)

bool DList::eraseFirstBiggerThan(int val);

This function removes the first node in the list that is bigger than the number val. If no nodes in list are bigger than val, function does not modify the linked list. Function returns true if a node was erased, false otherwise

template <typename T>

bool Sentinel<T>::eraseFirstBiggerThan(int val) {

//Enter code here for the sentinel version

std::cout << "Erase First Bigger Than code" << std::endl;

std::cout << "Val is: " << val << std::endl;

Node\* temp = front\_->next\_;

while (temp->data\_ > val) {

Node\* rm = temp;

rm->prev\_->next\_ = rm->next\_;

rm->next\_->prev\_ = rm->prev\_;

temp = temp->next\_;

delete rm;

return true;

}

return false;

}