Practice Final

Introduction to Algorithms & Programming II

COMP-1410 Summer (6-week)

Saturday August 17, 2024, 7:00pm

University of Windsor
School of Computer Science
Ryan Bluteau

READ ALL INSTRUCTIONS CAREFULLY.

1. **THIS IS NOT AN EXAM.** This is a practice final.
2. This is a closed book test. No laptops, no books, no phones, no calculators, no additional paper. Only a pen/pencil, eraser, and your ID should be with you.
3. Your full name and student ID number should be printed clearly on this page where indicated below.
4. Do not remove any pages from this exam. Missing pages will void your exam and will result in a grade of 0 for this exam.
5. You are not permitted to communicate with other students/people aside from the instructor or GA/TAs for the course. You cannot receive any unauthorized help with this exam. You cannot ask the GA/TAs for answers to an exam question. Otherwise, as per the senate bylaws, you will be reported and the exam will be voided.
6. If multiple answers are provided to a question, the worst grade between all answers is the final grade for that question. Multiple choice and fill-in-the-blanks style questions receive 0 when provided multiple answers.
7. Unclear answers will receive partial or complete mark deductions depending on severity.
8. The exam is 3 hours starting at 7:00pm. There are 60 marks total.
9. Good luck! Make sure to attempt all questions.

STATEMENT ON ACADEMIC INTEGRITY

I attest that all work in this exam is my own and completed independently without unauthorized help.	
First name:	
Last name:	
Student ID:	

Grades [Section for graders only]:

Question 1	Question 2	Question 3	Question 4	Question 5	Total
/ 10	**/ 10**	**/ 10**	**/ 10**	**/ 20**	**/ 60**

Question 1: Strings (you can use string.h). [10 marks – 2 marks each]

1A. Initialize a string containing the word Cat 4 different ways using one statement.

```
a. char str[4] = "Cat\0";

b. char str2[4] = "Cat";

c. char str3[4] = {'C', 'a', 't', '\0'};

d. char str4[4] = {'C', 'a', 't', 0};
```

1B. You are given the string variable str. Create a copy of this in 1 line.

```
char * copy_str = strdup(str); //need to be freed
```

1C. Create a character array containing the word Cat.

```
char char_arr[3] = {'C', 'a', 't'};
```

1D. You have a string "str" containing the word 'Con' and additional space up to 12 characters. You also have a character array "array" containing 'catenate'. We want to 'Concatenate' these in proper order (to form the word concatenate) and save the result in the original string variable "str". You have maximum three statements.

```
for(int k = 0; k < 8; k++)
    str[3 + k] = array[k];
str[11] = '\0';
```

1E. Calculate the length of a string using pointer notation**. You can't string.h for this question**.

```
int my_strlen(char *str){
     //int count = 0;
     //for(count = 0; str[count] != '\0'; count++)
     //   ;
     //return count;

     char *p = str;
     while(*p) p++;
     return p - str;

}
```

Question 2: Short answers. [10 marks – 2 marks each]

2A. What is the time complexity in terms of big O of the following algorithms: bubble sort, binary search, quick sort, merge sort.

O(n^2), O(logn), O(n^2), O(nlogn)

2B. Why would we still use quick sort if there exist algorithms with better time complexities? Please keep the answer short and to the point.

Average complexity of quicksort is O(nlogn) -> competes with mergesort
Merger sort consumes a lot of memory

2C. Create a pointer to store integers. Assign the pointer enough memory to fit 100 elements from the heap. Do this in 1 statement.

```
int *array = malloc(sizeof(int)*100); //-> heap
```
//int array[100]; -> stack (if in a function)

2D. In one line, open an existing file named "file.txt" with the intention to append more content.

FILE * fp = fopen("file.txt", "a");

2E. Calculate the complexity of the code below. You must use summation notation and show the steps to simplify it to big O notation.

(1)
```
char * str[] = "Some string with length N";
for(int i = 0; i < strlen(str); i++)
    if(str[i] >= 'A' && str[i] <= 'Z')
        str[i] = str[i] – 'A' + 'a';
```

$$\sum_{i=0}^{n-1}(O(n)+O(1)) = n(O(n)) = O(n^2)$$

(2)
```
char * str[] = "Some string with length N";
for(char *j = str; *j != '\0'; j++)
    if( *j >= 'a' && *j <= 'z')
        *j = *j – 'a' + 'A';
```

$$\sum_{j=0}^{n}(O(1)) = O(n)$$

Question 3: Diagrams. [10 marks – 5 marks each]

3A: Illustrate how bubble sort would sort the following array. It is expected that you show each step of the process by recreating the array after each swap. Any comparison made should underline the two values being compared. Using an X near the underline if no swap and a bidirectional arrow pointing at both elements to indicate the values should be swapped.

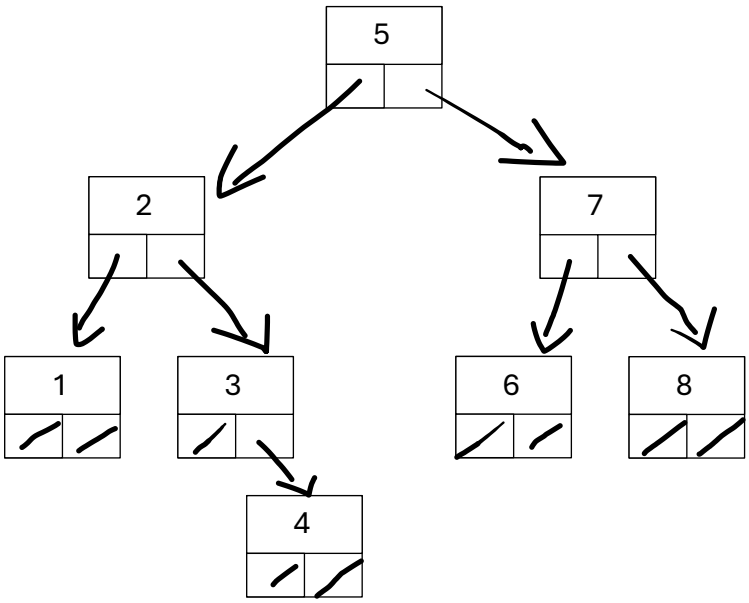
12	5	-2	-10	9	7	1

Example first step:

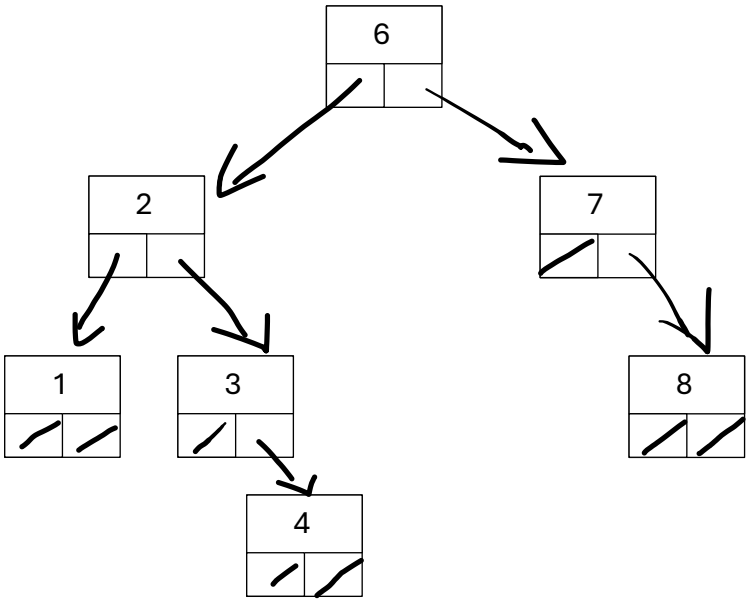
12	5	-2	-10	9	7	1	
5	**12**	**-2**	-10	9	7	1	//use bold for swap
5	-2	**12**	**-10**	9	7	1	
5	-2	-10	**12**	**9**	7	1	
5	-2	-10	9	**12**	**7**	1	
5	-2	-10	9	7	**12**	**1**	
5	**-2**	-10	9	7	1	12	
-2	**5**	**-10**	9	7	1	12	
-2	-10	5	9	7	1	12	
-2	-10	5	**9**	**7**	1	12	
-2	-10	5	7	**9**	**1**	12	
-2	**-10**	5	7	1	9	12	
-10	-2	5	7	1	9	12	
-10	-2	5	7	1	9	12	
-10	-2	5	**7**	**1**	9	12	
-10	-2	5	1	7	9	12	
-10	-2	5	1	7	9	12	
-10	-2	**5**	**1**	7	9	12	
-10	-2	1	5	7	9	12	
-10	-2	1	5	7	9	12	
-10	-2	1	5	7	9	12	
-10	-2	1	5	7	9	12	

3B: Given the binary search tree template below, do the following:

(1) Insert values (integers) and connect the tree to maintain its structure. The values can be anything, however must abide by the binary search requirements.



(2) Draw the resulting tree when the root is removed. Ensure the structure is maintained. Show how the values of your new tree change.



Question 4: Functions. [10 marks – 5 marks each]

4A. Recreate the strdup function from string.h.
// don't do this -> char * strcpy(char *dest, char *src);

```
char * strdup(char *str) {
        int len = strlen(str);
        char *dest = malloc(sizeof(char)*len + 1);
        strcpy(dest, str);
        return dest;
}
```

4B. Create a bubble sort algorithm to sort strings. The function is provided an array of strings and the length of that array.

```
{  "Cat",
   "Hello"
}

{  {'C', 'a', 't', '\0'}, -> constant strings (or same-length strings)
   {'H', 'e', 'l', 'l', 'o', '\0'}
}

void swap(char *a, char *b) {
        char temp = *a;
        *a = *b;
        *b = temp;
}

#define N 10
void bubble_sort(char *array[N], int len){
        for(int k = 0; k < len; k++){
                for(int j = 0; j < len – k – 1; j++) {
                        if(strcmp(array[j], array[j+1]) > 0) {
                                for(int i = 0; i < N; i++)
                                        swap(&array[j][i], &array[j+1][i]);
                        }
                }
        }
}
```

Question 5: Long answer. [20 marks – 5 marks each]

5A. Create the structure to handle a doubly linked list. Ensure you have a proper wrapper.

```
struct node {
        int item;
        struct node * next;
        struct node * prev;
};

struct list {
        struct node *front;
        struct node *tail;
};
```

5B. Create the init functions, init_dllist and init_node to create a doubly linked list and a new node.

```
struct list * init_dllist() {
        struct list * L = malloc(sizeof(struct list));
        if(L == NULL)
                return NULL;
        L->front = NULL;
        L->tail = NULL;
        return L;
}

struct node * init_node(int item) {
        struct node * new_node = malloc(sizeof(struct node));
        if(new_node == NULL)
                return NULL;
        new_node->item = item;
        new_node->next = NULL;
        new_node->prev = NULL;
        return new_node;
}
```

5C. Create the function dllist_insert to insert node in the list. It should maintain a sorted order.

```
//Return false on failure (malloc failed)
bool dllist_insert(int item, struct list *L) {
	//(1) Is the front empty?
	//	Insert at the front
	if(L->front == NULL) {
		L->front = init_node(item);
		if(L->front == NULL)
			return false;
		L->tail = L->front;
		return true;
	}
	//(2) Is the item the smallest item?
	//	Insert at the front
	if(L->front->item > item) {
		struct node * new_node = init_node(item);
		if(new_node == NULL)
			return false;
		struct node *temp = L->front;
		L->front = new_node;
		new_node->next = temp;
		temp->prev = L->front;
		return true;
	}
	//(3) Is the item being inserted in the middle?
	//	Insert between nodes
	struct node *pos = L->front;
	while(pos->next && pos->next->item < item)
		pos = pos->next;

	//4		5			6 || NULL
	//[pos]		[new node]		[???]
```

```
        //Skip if item exists
        if(pos->item == item)
                return true;

        struct node *new_node = init_node(item);
        if(new_node == NULL)
                return false;
        struct node *nextnext = pos->next;
        pos->next = new_node;
        new_node->next = nextnext;
        new_node->prev = pos;


        //(4) Is this the largest item>
        //        Place in tail
        if(nextnext != NULL) {
                nextnext->prev = new_node;
        } else {
                L->tail = new_node;
        }
        return true;
}
```

5D. Create the function dllist_destroy to destroy the list. *[Hint: Make it recursive.]*

```
void destroy_nodes(struct node * N) {
	if(N) {
		destroy_nodes(N->next);
		free(N);
	}
}

void dllist_destroy(struct list * L) {
	destroy(L->front);
	free(L);
}
```