CMPT 125 D200, Spring 2026

Midterm Exam - SOLUTIONS
March 6, 2026

Name_____

SFU ID: |__|__|__|__|__|__|__|__|__|

Problem 1	
Problem 2	
Problem 3	
Problem 4	
TOTAL	

Instructions:

1. The duration of the exam is 100 minutes.
2. Write your full name and SFU ID **clearly**.
3. This is a closed book exam, no calculators, cell phones, or any other material.
4. The exam consists of four (4) problems.
5. Write your answers in the provided space.
6. There is an extra page at the end of the exam. You may use it if needed.
7. **Explain all your answers.**
8. **Really, explain all your answers.**

Good luck!

Problem 1 [25 points]

a) [2 points each] Suppose you have a program in C named *program.c*.

1. What is the command in Linux to compile it into an executable file?

   Answer: gcc program.c

2. What is the default name of the compiled executable in Linux?

   Answer: a.out

3. How can you specify the name of the executable in the compilation line?

   Answer: gcc program.c -o specific_name

b) [7 points] Will the code below compile? If yes, what will be the result of the execution? If there are any errors/warnings/potential issues explain them.

```
#include <stdio.h>

enum suit {CLUBS, DIAMONDS, HEARTS, SPADES};

int fun(int* x, int* y) {
      int z = 4;
      *y = z;
      *x = HEARTS+1;
      y = z;
      *y = 8;
      return z;
}

int main() {
      int a = CLUBS, b = DIAMONDS;
      int c = fun(&a, &b);
      printf("a = %d, b = %d, c = %d\n", a, b, c);
      return 0;
}
```

z = 4
*y = 4 → main.b = 4
*x = 3 → main.a = 3
incompatible types - y points to address 4
will probably crash here
if didn't crash, returns 4

a = 0, b = 1

Answer: Probably there will be a warning because of the line y = z (assign int to int*)
It will probably crash because when trying to assign *y=8
If it doesn't crash, it will print: a = 3, b = 4, c = 4

c) [6 points] Fill in the blanks in the code, so that the function implements strcat().
The function takes two strings dest, and src, and appends src to the end of dest.
The function returns the pointer to dest.

For example, if `dest` is the string `"ABC"` and `src` is the string `"1234"`, then after the function returns, `dest` becomes `"ABC1234"`.

You may use standard library functions. Explain your answer.

```
#include <string.h>

chat* strcat(char* dest, const char* src) {


    return strcpy(__dest + strlen(dest)___,__src__);
}
```

d) [6 points] Implement the function strdup(). The function gets a string, and returns a new copy of the same string.
The program below needs to print `"Hello"` without an error message.

You may not use the string.h library.

```
#include <stdio.h>
#include <stdlib.h>

char* strdup(char* src) {
  char* ret = (char*) malloc(strlen(src)+ 1); // +1 for '\0' at the end
  if (ret == NULL) return NULL;
  int i;
  for(i = 0; src[i] != '\0'; i++) {
    ret[i] = src[i];
  ret[i] = '\0';
  return ret;
}
int main() {
  char* str = "Hello";
  char* s_copy = strdup(str);
  if (str==s_copy)
    printf("ERROR: same string\n");
  printf("%s\n", s_copy);
  free(s_copy);
  return 0;
}
```

Problem 2 [25 points]

a) [5 points] Write a function that gets an array of ints of length n, and a boolean predicate pred. It returns the number of indices i such that pred(A[i])==true.

```
int count(int* A, int n, bool (*pred)(int)) {

  int count = 0;
  for (int i=0; i<n; i++) {
    if (pred(A[i]))
      count++;









}
```

b) [5 points] Write a function that gets an array of ints of length n, and a function f.
It applies f on each element of the array.

```
void map(int* A, int n, int (*f)(int)) {



  for (int i=0; i<n; i++) {
    A[i] = f(A[i]);









}
```

c) [15 points] Write a function that gets a string, and computes the length of the longest substring that is a palindrome.
For example,

- `longest_substring_palindrome("abEYE123")` needs to return 3.
- `longest_substring_palindrome("12345")` needs to return 1.
- `longest_substring_palindrome("")` needs to return 0.
- `longest_substring_palindrome("a228822A")` needs to return 6.

The function does not need to be the most efficient.

```
int longest_substring_palindrome(const char* str) {


  int n = istrlen(str);
  int max = 0;

  // we check all substrings str[i…j] if they are palindromes
  // and return the length of the longest one
  // the length of str[i…j] is j-i+1
  for (int i=0; i<n; i++) {
    for (int j=i+max; j<n; j++) { // we check only j>i+count
      if (is_substring_palindrome(str, i, j)
        max = j-i+1;

  return max;

}
```

We use the following helper function

```
bool is_substring_palindrome(const char* str, int i, int j) {

  int mid = (j-i)/2;

  for (int k=0; k<=mid; k++)
    if (str[i+k] != str[j-k])
      return false;

  return true;

}
```

Problem 3 [25 points]

a) [8 points] Recall the **MergeSort** algorithm.

```
void merge_sort(int* A, int n) {
  if (n >= 2) {
    int mid = n/2;
    merge_sort(A, mid);          // sort left half
    merge_sort(A+mid, n-mid);    // sort right half
    merge(A,n,mid); // merge() we saw in class with running time O(n)
  }
}
```

What is the running time of Merge Sort, when applied on the array [0,1,2,3,...n-2,n-1]? Use big-O notation to express your answer. Write the tightest possible bound on the running time. Explain your answer.

Answers: The running time is O(nlog(n)).
If we denote by T(n) the running time of merge sort on a sorted array, then we get the recursive formula $T(n) = 2*T(n/2) + O(n)$

- $T(n/2)$ are the recursive calls
- $O(n)$ is the running time of merge()

We saw in class that the solution to this recursive definition is O(n log(n))

b) [7 points] Give an example of an array of length 7, on which **InsertionSort** makes a total of exactly 20 swaps.

Answers: A = [7,6,5,4,3,1,2]
Below we list the number of swaps in each iteration:

- Insert 7: 0 swaps → A = [7,6,5,4,3,1,2]
- Insert 6: 1 swaps → A = [6,7,5,4,3,1,2]
- Insert 5: 2 swaps → A = [5,6,7,4,3,1,2]
- Insert 4: 3 swaps → A = [4,5,6,7,3,1,2]
- Insert 3: 4 swaps → A = [3,4,5,6,7,1,2]
- Insert 1: 5 swaps → A = [1,3,4,5,6,7,2]
- Insert 2: 5 swaps → A = [1,2,3,4,5,6,7]

Total: 1+2+3+4+5+5 = 20 swaps

c) [10 points] Write a sorting function that gets an array of length n, consisting of only the numbers 1,2,3,4,5 and sort the array in O(n) time.
For example, if the given array is [1,4,2,4,2,2,5,1], then the function changes to arrays to [1,1,2,2,2,4,4,5]

```
void sort12345(int* a, int n) {
```

Idea: we count in O(n) time how many ones we have, how many twos… up to five.
Then start populating the array by putting count[1] many ones, then count[2] many twos, then count[3] many threes and so on

```
  // count[i] will contain the number of i's in the array
  // count[0] remains 0
  int count[6] = {0,0,0,0,0,0};
  for (int i = 0; i < n; i++)
    count[a[i]]++;

  // we add to array count[1] many ones
  // then we add count[2] many twos
  // and so on un to 5
  int j=0; // j will be the index in the array we are populating
  for (int digit = 1; digit<=5; digit++)
    for (int i = 0; i < count[digit]; i++) {
      a[j] = 1;
      j++;
  }
}
```

Problem 4 [25 points]

Consider the following function

```
bool foo(int* a, int n) {
  if (n <= 1)
    return true;

  int mid = n/2; // round down if len is odd
  bool b1 = foo(a, mid);
  bool b2 = foo(a+mid, n-mid);
  return (b1 && b2 && a[mid-1] <= a[mid]);
}
```

a) [7 points] Explain in simple words the functionality of foo().

Answer: The function checks that the array is sorted.
That is, it returns true if a[i-1]<a[i] for all i=1…n-1, and returns false otherwise

This can be seen by the recursion.
The function checks (last) that a[mid-1]<=a[mid], and then
recursively check the two halves: a[0…mid-1] and a[mid…n-1]

b) [8 points] What is the running time of foo()? Use big-O notation to express your answer.

Answer: The running time is O(n).

Not that each index i=1…n-1 appears as mid exactly once. And for each such mid we perform O(1) operations + 2 recursive calls.

Alternatively, we can denote the running time by T(n).
Then T(1)= 1, and T(n) = 2*T(n/2) + C for some constant C. Let's say C=5.

It is not difficult to show, e.g. by induction that T(n) < 2C*n

c) [10 points] Write a **recursive** binary search. The function gets a sorted array A of length n, and elt, and returns an index i such that A[i]==elt.
If elt appears in A more than once, you may return any index i such that A[i]=elt.
If elt is not in the array, the function returns -1.

A solution with a helper function will get a maximum of 7 points.
For full marks solve the problem without a helper function.

The function needs to run in O(log(n)) times.

```
int binary_search_rec(const int* A, int n, int elt) {

  // base case
  if (n==0)
    return -1;
  if (n==1)
    if (A[0]==elt)
      return 0;
    else
      return -1;


  // inductive step
  int mid = n/2;
  if (A[mid]==elt)
    return mid;
  if (elt < A[mid]) { // go left
    return binary_search_rec(A, mid, elt);
  }
  else { // go right + add mid to the result
    int relative_ind = binary_search_rec(A+mid, n-mid, elt);
    if (relative_ind==-1)
      return -1;
    else
      return mid + relative_ind;
  }




}
```

Extra page

Empty page