CMPT 125 D200, Spring 2022

Midterm Exam - SOLUTIONS
February 28, 2022

Name_____

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

Problem 1	
Problem 2	
Problem 3	
Problem 4	
TOTAL	

Instructions:

1. 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. Each problem is worth 25 points.
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) [6 points] What will be the output of the following program? Explain your answer.

```
#include <stdio.h>
enum emph {BOLD, ITALIC, UNDERLINE};

int foo(int x, int *y) {
     int* z = &x;
     *y = *z;
     *z = -1;
     return x;
}
int main() {
     int a = BOLD, b = 1, c = 2;
     c = foo(a, &b);
     printf("a = %d, b = %d, c = %d", a, b, c);
     return 0;
}
```

ANSWER: a = 0 b = 0 c = -1

Explanation: In the beginning a=0, b=1, c=2

- x gets the value 0 (unrelated to a), y is pointer to b.
- The line z=&x: makes z point to x. Hence *z=0
- *y = *z = 0, This sets **b=0**
- *z = -1 sets x=-1. Therefore the function returns -1, and **c is assigned -1**

b) [6 points] Will the code below compile?
If yes, what will be the output? If not, explain why.

```
#include <stdio.h>

int main() {
    char s1[20] = {'A','B',0,'C','D','E',0};
    char* str = s1;
    while (*str) {
      str = str+1;
    }
    printf("%s\n", str+1);
    return 0;

}
```

ANSWER: the function will print “CDE”

Explanation: The loop will stop at the first 0 (after AB).
And str+1 points to ‘C’

c) [7 points] Will the code below compile?
If yes, what will be the output? If not, explain why.

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

int* get_arr() {
  int* arr = malloc(3*sizeof(int));
  int* ret = arr;
  arr[0]=2;
  arr[1]=3;
  ret[2]=6;
  return ret;
}

int main() {
  int* a1 = get_arr();
  a1[0]=5;
  printf("a1 = [%d, %d, %d]\n", a1[0], a1[1], a1[2]);
  return 0;
}
```

ANSWER: all will compile ok, and will print: a1 = [5, 3, 6]
Explanation: the array returned will have values [2,3,6]
a1 points to this array, and sets a1[0]=5.
Pretty straightforward, no tricks.

d) [6 points] Let T(n) be the running time of foo(n). Use Big-O notation to express T(n).
Explain your solution.

```
void foo(int n) {
  if (n>0) {
    int third = n/3; // if n is not divisible by 3, n/3 is rounded down
    foo(third);
    for(int i=third+1; i<=n ; i++)
      printf("i = %d ", i);
    printf("\n");
  }
}
```

ANSWER: The running time is essentially equal to the number of times printf is called.
Observe that the function called foo(third), and then prints the number third+1…n
The recursive calls print 1…third (in some order with '\n' between them.

Overall, each number between 1 and n is printed once, and the total running time is O(n).

Problem 2 [25 points]

a) [5 points] Consider the **Binary Search** algorithm. How many comparisons will it make on the input A = [1, 5, 7, 8, 20, 25, 30, 40, 60, 61, 62] when searching for 8? Explain your answer.

ANSWER1: 3 comparisons
We first compare 8 to 25 and go left to [1,5,7,8,20]
Then, we compare 8 to 7, and to right to [8,20]
Then, we compare 8 to 8 and declare "FOUND"
Total 3 comparisons

ANSWER2: 4 comparisons
We first compare 8 to 25 and go left to [1,5,7,8,20]
Then, we compare 8 to 7, and to right to [8,20]
Then, we compare 8 to 20 and go left to [8]
Then, we compare 8 to 8 and declare "FOUND"
Total 4 comparisons

b) [8 points] Show an array with the values {1, 2, 3, 4, 5, 6, 7, 8} so that the **SelectionSort** makes swaps only in the last three iterations of the outer loop, and no other swaps.

ANSWER2:Selection sort on an array of length 8 has 7 outer iterations.
The first four need not swap anything
The remaining three iterations need to make a swap:

Since the first 4 iterations don't make swaps, we put in the beginning the small numbers:[1,2,3,4]

A bit of trial and error can give for example: [1,2,3,4,7,5,8,6]
First four iteration keep the array as is
Iteration 5: swaps 5 with 7 → [1,2,3,4,5,7,8,6]
Iteration 6: swaps 6 with 7 → [1,2,3,4,5,6,8,7]
Iteration 7: swaps 7 with 8 → [1,2,3,4,5,6,7,8]

Another solution is [1,2,3,4,6,7,8,5]

c) [12 points] Implement the merge function that gets an array A of length n, and index mid, such that A[0,...mid-1] and A[mid…n-1] are sorted in the increasing order.
The function merges the two halves of A into a sorted array in time O(n).
* Note that some elements might be equal.
Remember to use malloc/free if you need to use a new array.

Explain your code if necessary.

```
void merge(int* A, int n, int mid) {
  int* tmp = (int*)malloc(n*sizeof(int)); // tmp collects all numbers.
                                          // to be released at the end
  int ptr1 = 0, ptr2 = mid; // two pointers to the two halves of A
  int next_ind = 0; // stores the next index in tmp where a new value will be
added
  // copy the minimal
  while(ptr1<mid && ptr2<n) {
    if (A[ptr1] < A[ptr2]) {  // move up ptr1
      tmp[next_ind] = A[ptr1];
      ptr1++;
    }
    else {  // move up ptr2
      tmp[next_ind] = A[ptr2];
      ptr2++;
    }
    next_ind++; // don't forget to to up next_int
  }
  if (ptr1==mid) { // copy the remainder of A[ptr2…n-1]
      while (ptr2<n) {
      tmp[next_ind] = A[ptr2];
      ptr2++;     next_ind++;
      }
  }
  else { // ptr2==n  -- copy the remainder of A[ptr1…mid-1]
      while (ptr1<mid) {
      tmp[next_ind] = A[ptr1];
      ptr1++;     next_ind++;
      }
  }
  // copy from tmp to A and release tmp
  for (int i=0;i<n;i++)
      A[i] = tmp[i];
  free(tmp);// don't forget to free tmp
}
```

Problem 3 [25 points]

a) [8 points] Write a function that gets two strings and checks if str1 is the prefix of str2.

- is_prefix("abcd", "abcdef") returns true.
- is_prefix("a12b", "a12b") returns true.
- is_prefix("abcd", "ab") returns false.
- is_prefix("abcd", "KLM") returns false.

Explain your idea before writing code.

```
bool is_prefix(const char* str1, const char* str2) {


  while (*str1) {
      if (*str1 != *str2)
      return false;
      str1++;
      str2++;
  }
  return true;

}
```

```
}
```

[4 points] What is the running time of your function? Use big-O notation to state your answer. Give the tightest possible answer.

ANSWER: The function has a while loop that runs until the end of str1, and in each iteration compares the two pointers and increments them. Therefore, the total running time is O(strlen(str1))

b) [10 points] Implement the function str_find that gets two strings, *text* and *pattern*, and searches for the *pattern* as a substring of *text*. This function returns the index representing the beginning of the first appearance of the *pattern* in the *text*.
If *pattern* is not a substring of *text*, the function returns -1.
For example

- str_find("Hel**lo w**orld, Hello", "lo w") returns 3.
- str_find("aBaB**aBC**DaBC", "aBC") returns 4.
- str_find("Hello world", "wrd") returns -1.

You are not allowed to use any library functions to solve this, except for strlen().

```
int str_find(const char* text, const char* pattern) {
  // check in each iteration text[i...]starts with pattern
  int i = 0;
  while (*(text+i)) {
      // check is text[i...] starts with the pattern
      // if yes, return i
     if (is_prefix(pattern, text+i))
       return i;

      i++; // otherwise, increment i
  }
  return -1;
}
```

```
}
```

[3 points] What is the running time of your function in terms of the length of the strings in the worst case? Use big-O notation to state your answer. Give the tightest possible answer.

ANSWER: Let n = strlen(text) and k = strlen(pattern).
The outer loop performs at most n iterations.
In each iteration it calls is_prefix, which runs in O(k) time.
Therefore, the total running time is O(nk)

Problem 4 [25 points]

Consider the following function.

```
int what(unsigned int n) {
  if (n<=2)
    return n;
  int sum=0;
  for(int i=n/2; i<n; i++) // if n is odd, then n/2 is rounded down
    sum = sum+what(i);
  return sum;
}
```

a) [4 points] Compute what(4). Explain your answer.

ANSWER:
By the stopping condition of the induction we have

- what(1) = 1
- what(2) = 2

Next we compute what(3):

- what(3) = what(1) + what(2) = 3

Next we compute what(4):

- what(4) = what(2) + what(3) = 2+3 = 5

b) [9 points] Rewrite the function what() with the same functionality so that on input n, it returns the answer in time $O(n^2)$. Explain your answer.

```
int what(unsigned int n) {
  if (n<=2)
     return n;
  // w[k] will store what(k)
  int* w = (int*)malloc((n+1)*sizeof(int));
  w[0] =0; w[1] = 1; w[2] = 2;
  for(int k=3; k<=n; k++) { // we compute w[k] in the array
     // compute w[k] using w[0…k-1] (instead of recursion
     int sum=0;
     for(int i=k/2; i<k; i++)
        sum = sum+w[i];
  }
  int ret = w[n];
  free(w);
  return ret;

}
```

c) [12 points] Write a function that gets an array of ints of length n, and returns an array of length n, such that output[i] is equal to the sum of the elements in the input subarray [0... i]. For example,

- input [1, 4, 3, 8, 2, 9]
- output [1, 5, 8, 16, 18, 27].

Make sure the returned array is allocated on the heap.

You may write helper functions if that makes the solution more readable.

- A correct answer with linear running time, will give you 15 points
- A correct answer with quadratic running time, will give you 10 points

```
int* sum_prefixes(const int* ar, int n) {


  if (n==0)
      return NULL; // not important. We can assume that n>0

  int* ret = (int*)malloc(n*sizeof(int));
  ret[0] = ar[0];
  for(int i=1;i<n;i++) {
      // note that sum ar[0…i] can be obtained from ret[i-1]+ar[i]
      ret[i] = ret[i-1]+ar[i];
  }
  return ret;









}
```

Extra page

Empty page