CMPT 125 D100, Spring 2022

Midterm Exam - Solutions
March 4, 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 colors {RED, GREEN, BLUE, YELLOW};

void foo(int* x, int* y) {
      long z = 3;
      *y = 6;
      x = y;
      *x = z;
}

int main() {
      int a = GREEN, b = YELLOW;
      foo(&a, &b);
      printf("a = %d, b = %d", a, b);
      return 0;
}
```

ANSWER: a = 1 b = 3

Explanation: In the beginning a=1, b=3

- x is a pointer to a, y is a pointer to b.
- *y=6 sets **b=6**
- x=y sets x to point to b (and x doesn't point to a anymore)
- *x = z sets **b=z=3**

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 s[10] = {'A','B',0,'C','D','E','F','G',0};
    int what=0;
    while (str[what]) {
      what = what+1;
    }
    printf("%d\n", what);
    return 0;
}
```

ANSWER: the function will print 2

Explanation: The loop will stop when str[what]=0 for the first time. This happens when what=2.

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

```
#include <stdio.h>

int* bar_arr() {
  int arr[4];
  for(int i=0;i<4;i++)
    arr[i]=1;
  int* ret = arr;
  return ret;
}

int main() {
  int* a1 = bar_arr();
  a1[0] = 2;
  printf("a1 = [%d, %d, %d, %d]\n", a1[0], a1[1], a1[2], a1[3]);
  return 0;
}
```

ANSWER: the function will compile.
The function returns the array with values [1,1,1,1], which is assigned to a1.
Then, in main we change a1[0] = 2, and print “a1 = [2, 1, 1, 1]”

d) [6 points] Let T(n) be the running time of foo(1,n). Use Big-O notation to express T(n). Explain your solution.

```
void foo(int k, int n) {
  if (k<=n) {
    int mid = (k+n)/2; // if (k+n) is odd, (k+n)/2 is rounded down
    for(int i=k; i<=mid ; i++)
      printf("i = %d  ", i);
    printf("\n");
    foo(mid+1, n);
  }
}
```

ANSWER: The running time is essentially equal to the number of times printf is called.
foo(1,n) sets mid = (n+1)/2 and prints [1…mid]
And then makes a recursive call foo(mid+1,n).
This recursive call prints **the first half** of the interval [mid+1…n], and makes recursive call onthe other half
and so …
Overall, each number between 1 and n is printed once, and the total running time is O(n).

Alternative solution: The recursive formula for running time is T(n)=C*n + T(n/2)
If we open this we get:
T(n) = C*n + T(n/2) = C*n + C*n/2 + T(n/4) = Cn + C*n/2 + C*n/4 + T(n/8)... = O(n)

Problem 2 [25 points]

a) [5 points] Consider the **Binary Search** algorithm. How many comparisons will it make on the input A = [2, 4, 6, 8, 9, 12, 14, 15, 18, 90, 99] when searching for 15? Explain your answer.

ANSWER1: 3 comparisons
We first compare 15 to 12 and go right to [14, 15, 18, 90, 99]
Then, we compare 15 to 18, and to left to [14, 15]
Then, we compare 15 to 15 and declare “FOUND”
Total 3 comparisons

ANSWER2: 4 comparisons
We first compare 15 to 12 and go right to [14, 15, 18, 90, 99]
Then, we compare 15 to 18, and to left to [14, 15]
Then, we compare 14 to 15 and go right to [15]
Then, we compare 15 to 15 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 **InsertionSort** makes exactly 6 swaps in the last iteration of the outer loop, and makes no other swaps.

ANSWER:
The first 7 numbers must be sorted. The last number will be swapped 6 times.
The answer is [1,3,4,5,6,7,8,2]
2 will be swapped 6 times

c) [12 points] Implement the merge function that gets an array A of length n, and index mid, such that A[0,...mid] and A[mid+1…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 a string and computes the length of its longest prefix consisting only of the lowercase letters. For example,

- longest_lower_case_prefix("abCDef") is 2 - the prefix is "ab"
- longest_lower_case_prefix("12abcd") is 0 - the string starts with "12"
- longest_lower_case_prefix("abc") is 3 - the prefix is "abc"

Explain your idea before writing code.

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

  int ret = 0;
  while (str[ret] >= 'a' && str[ret <='z'])
       ret++;
  return ret;
}
```

```
}
```

[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 number of iterations is equal to the returned answer (length of its longest prefix consisting only of the lowercase letters). Therefore, if the answer is k, then the running time is O(k)

** O(length of str) is also an acceptable answer, but it is not tight if k is much smaller than the length of str.

b) [10 points] Implement a function that gets a string *str*, and a positive integer k, and searches for a substring of str of length exactly k that is a palindrome.
The function returns the index in *str* representing the beginning of such a palindrome.
If *str* contains more than one such palindrome, the function returns the index to the first palindrome of length k.
If *str* does not contain a palindrome of length k, the function return -1
For example,

- find_k_palindrome("**ABBA**bcd", 4) returns 0.
- find_k_palindrome("Hell**owo**rld wowowow", 3) returns 4.
- find_k_palindrome("**wow**owowHelloworld", 3) returns 0.
- find_k_palindrome("D**ABCDCBA**", 7) returns 1.
- find_k_palindrome("Rolling Stones", 6) returns -1.

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

```
char* find_k_palindrome(const char* str, int k) {
// we use a helper function is_k_pal() defined below:
  int len = strlen(str);
  int start_ind = 0;
  while (start_ind <= len-k) {
      if (is_k_pal(str, start_ind, k))
            return start_ind;
      start_ind++;
  }
  return -1;
}

// helper function that checks if str[start_ind...start_ind+k-1] is a
palindrome
// assumption length of str is at least start_ind+k
bool is_k_pal(const char* str, int start_ind, int k) {
  int i=0;
  while (i<=k/2) {
      if (str[start_ind+i] != str[start_ind+k-i-1])
      return false;
      i++;
  }
  return true;
}
```

[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:
is_k_pal() has running time O(k)
If n is the length of str, then the outer loop has n iterations, each calling is_k_pal().
Therefore, the total running time is O(nk)
// O(n^2) will also be accepted if the explanation is correct

Problem 4 [25 points]

Consider the following function.

```
int what(unsigned int n) {
  if (n<=2)
    return n;
  return (n-1)*what(n-1) + (n-2)*what(n-2);
}
```

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) = 2*what(2) + 1*what(1) = 5

Next we compute what(4):

- what(4) = 3*what(3) + 2*what(2) = 3*5+2*2 = 19

b) [9 points] Rewrite the function what() with the same functionality so that on input n, it returns the answer in time O(n). 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 i=3; i<=n; i++) { // we compute w[i] in the array
       w[i] = (i-1)*w[i-1]+(i-2)*w[i-2];
  }
  int ret = w[n];
  free(w); // don't forget to free w
  return ret;
}
```

The running time is clearly O(n) - one loop with O(1) time in each iteration

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 maximal element in the input subarray [i…n-1].
For example,

- input [1, 4, 9, 8, 2, 5]
- output [9, 9, 9, 8, 5, 5].

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* max_suffix(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[n-1] = ar[n-1];
   for(int i=n-2;i>=0;i--) {
        if (ret[i+1]>ar[i])
                ret[i]=ret[i+1];
        else
                ret[i]=ar[i];
   }
   return ret;

}
```

Extra page

Empty page