Xi Liu, xl3504, Homework 6

Problem 1
assume $T(n) \geq c2^{n/2}$ for all positive $m < n$, $c \in \mathbb{R}^+$, in particular for $m = n-2$, the inductive hypothesis is $T(n-2) \geq c2^{(n-2)/4}$

$$\begin{aligned}
T(n) &= T(n-1) + T(n-2) + c \\
&= T(n-1) + T(n-2) + c \\
&/* \text{ since } T(n-1) \geq T(n-2) \ */ \\
&\geq T(n-2) + T(n-2) + c \\
&= 2T(n-2) + c \\
&\geq 2(c2^{(n-2)/4}) + c \\
&= \frac{2}{2^4} c2^{n-2} + c \\
&= \frac{2}{2^6} c2^n + c \\
&= \frac{c}{2^5} 2^n + c
\end{aligned}$$

$T(n) \geq \frac{c}{2^5} 2^n + c$, so $T(n) = \Omega(2^n) = \Omega(2^{n/2})$

Problem 2
(a)

```
/* to find the recursion,
the recursive version is included below */

#include <iostream>
#include <string.h>
using namespace std;

int lcs_rec(const char * a, const char * b, int i, int j, int cnt)
{
    if(!i || !j)
        return cnt;
    if(a[i - 1] == b[j - 1])
        cnt = lcs_rec(a, b, i - 1, j - 1, cnt + 1);
    cnt = max(cnt,
              max(lcs_rec(a, b, i - 1, j, 0),
              lcs_rec(a, b, i, j - 1, 0))
              );
    return cnt;
}
int main()
{
    const char * a = ...
    const char * b = ...
    cout << lcs_rec(a, b, strlen(a), strlen(b), 0);
}
```

so the recursion is

$$lcs(i,j,cnt) = \begin{cases} cnt & \text{if } i = 0 \text{ or } j = 0 \\ lcs(i-1, j-1, cnt+1) & \text{if } a[i-1] = b[j-1] \\ max(cnt, lcs(i-1,j,0), lcs(i,j-1,0)) & \text{if } a[i-1] \neq b[j-1] \end{cases}$$

(b)
if $n = strlen(S) = 0$ or $m = strlen(T) = 0$, then $lcs(i, j, cnt) = cnt = 0$

(c)

```
#include <stdio.h>
#include <string.h>
#include <math.h>

int lcs(char * a, int m, char * b, int n)
{
    int memo[m + 1][n + 1];
    int ret = 0;
    for(int i = 0; i <= m; ++i)
    {
        for(int j = 0; j <= n; ++j)
        {
            if(!i || !j)
                memo[i][j] = 0;
            else if(a[i - 1] == b[j - 1])
            {
                memo[i][j] = memo[i - 1][j - 1] + 1;
                ret = fmax(ret, memo[i][j]);
            }
            else
                memo[i][j] = 0;
        }
    }
    return ret;
}

int main()
{
   char * a = ...
   char * b = ...
   printf("%d\n", lcs(a, strlen(a), b, strlen(b)));
}
```

(d)
time complexity = $\Theta(mn)$
2 levels of for loop
operations within the innermost level are constant

Problem 3

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

int coin(int *** m)
{
    int ** memo = malloc(n * sizeof(int *));
    *m = memo;
    for(int i = 0; i < n; ++i)
    {
        size_t sz = n * sizeof(int);
        memo[i] = malloc(sz);
        memset(memo[i], 0, sz);
    }
    for(int diag = 0; diag < n; ++diag)
    {
        for(int i = 0, j = diag; j < n; ++i, ++j)
        {/* only use values from upper right triangular matrix */
            int a1 = (i + 2) <= j ? memo[i + 2][j] : 0;
            int a2 = ((i + 1) <= (j - 1)) ? memo[i + 1][j - 1] : 0;
            int a3 = (i <= (j - 2)) ? memo[i][j - 2] : 0;
            memo[i][j] = fmax(a[i] + fmin(a1, a2),
                              a[j] + fmin(a2, a3));
        }
    }
    return memo[0][n - 1];
```

```
}

int main()
{
    int ** m;
    printf("coin = %d\n", coin(&m));
}
```

time complexity = $\Theta(n^2)$

Problem 4
(a)

$$P(i,j) = \begin{cases} 1 & \text{if } i = j \\ P(i+1, j-1) + 2 & \text{if } S[i] = S[j] \\ max(P(i+1,j), P(i,j-1)) & \text{if } S[i] \neq S[j] \end{cases}$$

(b) base cases: a subsequence of 1 character is a palindrome of length 1
so if $i = j, \quad P(i,j) = 1$

(c)

```
#include <stdlib.h>
#include <string.h>
#include <math.h>

int longest_palin_subseq(char * a)
{/* consecutive not required */
    int n = strlen(a);
    int memo[n][n];
    memset(memo, 0, sizeof(memo));
    for(int i = 0; i < n; ++i)
        memo[i][i] = 1;
    for(int sub_len = 2; sub_len <= n; ++sub_len)
    {
        for(int i = 0; i < n - sub_len + 1; ++i)
        {
            int j = i + sub_len - 1;
            if(a[i] == a[j])
                memo[i][j] = memo[i + 1][j - 1] + 2;
            else
                memo[i][j] = fmax(memo[i + 1][j], memo[i][j - 1]);
        }
    }
    return memo[0][n - 1];
```

}

(d)
time complexity = $\Theta(n^2)$
2 levels of for loop
operations within the innermost level are constant

Problem 5

```
/* longest_palin_subseq2() computes
the length of the longest palindromic subsequence
using longest_common_subseq() */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

int longest_common_subseq(char * a, char * b)
{
    int a_len = strlen(a), b_len = strlen(b);
    int c[a_len + 1][b_len + 1];
    for(int i = 0; i <= a_len; ++i)
    {
        for(int j = 0; j <= b_len; ++j)
        {
            if(!i || !j)
                c[i][j] = 0;
            else if(a[i - 1] == b[j - 1])
                c[i][j] = c[i - 1][j - 1] + 1;
            else
                c[i][j] = fmax(c[i - 1][j], c[i][j - 1]);
        }
    }
    return c[a_len][b_len];
}

char * revstr(char * a)
{
    int a_len = strlen(a);
    char * rev = malloc((a_len + 1) * sizeof(char));
    strcpy(rev, a);
    for(int i = 0, j = a_len - 1; i < j; ++i, --j)
    {
        char t = rev[i];
```

```
        rev[i] = rev[j];
        rev[j] = t;
    }
    return rev;
}

int longest_palin_subseq2(char * a)
{
    char * rev = revstr(a);
    return longest_common_subseq(a, rev);
}
```

Problem 6

(a)

$$max\_reward(i,j) = \begin{cases} A[0][0] & \text{if } i = 0 \wedge j = 0 \\ A[i][j] + max(max\_reward(i-1,j), max\_reward(i,j+1)) & \text{else} \end{cases}$$

(b)

at the starting position $i = 0 \wedge j = 0$, only possible gold to be collected is $A[0][0]$

if $i = 0 \wedge j = 0$, $max\_reward(i,j) = A[0][0]$

(c)

```
int max_reward()
{ /* each entry in a[][] store the reward */
    int memo[m][n];
    memset(memo, 0, sizeof(memo));
    **memo = **a;
    for(int i = 0; i < m; ++i)
    {
        for(int j = 0; j < n; ++j)
        {
            if(!i && !j)
                continue;
            int a1 = (i - 1) >= 0 ? memo[i - 1][j] : 0;
            int a2 = (j - 1) >= 0 ? memo[i][j - 1] : 0;
            memo[i][j] = a[i][j] + fmax(a1, a2);
        }
    }
    return memo[m - 1][n - 1];
}
```

(d)
time complexity = $\Theta(mn)$
2 levels of for loop
operations within the innermost level are constant

Problem 7

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

int longest_inc_subseq(int * a, int n)
{
    int memo[n];
    for(int i = 0; i < n; ++i)
    {
        memo[i] = 1;
        for(int j = 0; j < i; ++j)
        {
            if(a[j] < a[i])
                memo[i] = fmax(memo[i], memo[j] + 1);
        }
    }
    return memo[n - 1];
}
```

time complexity = $\Theta(n^2)$