problem 1

Problem 1:

Ackermann's function $A(m,n)$ is defined as follows:

$$A(m,n) = \begin{cases} n+1 & \text{, if } m = 0 \\ A(m-1,\ 1) & \text{, if } n = 0 \\ A(m-1,\ A(m,n-1)) & \text{, otherwise} \end{cases}$$

This function is studied because it grows very fast for small values of m and n. Write a recursive function for computing this function. Then write a nonrecursive algorithm for computing Ackermann's function.

完整程式碼:

```
/*  1017 hw1 資工二乙 41143264 楊育哲
    A(m, n) 遞迴 (阿克曼函數)
*/
#include <iostream>
using namespace std;

int recA(int m, int n){//以if-else實作遞迴版本
    if(m==0) return n+1;
    else if(n==0) return recA(m-1, 1);
    else return recA(m-1, recA(m, n-1));
}

int nonrecA(int m, int n){//以stack實作蝶帶版本
    int stackOfM[100]={0}, current=0;
    stackOfM[0] = m;
    while(current>=0){
        m = stackOfM[current--];
        if(m==0) n++;
        else if(n==0){
            n=1;
            stackOfM[++current]=m-1;
        }else{
            stackOfM[++current]=m-1;
            stackOfM[++current]=m;
            n--;
        }
    }
    return n;
}

int main(){
    cout<<recA(1, 1)<<" "<<nonrecA(1, 1);
```

```
    return 0;
}
```

recursive function:

```
int recA(int m, int n){//以if-else實作遞迴版本
    if(m==0) return n+1;
    else if(n==0) return recA(m-1, 1);
    else return recA(m-1, recA(m, n-1));
}
```

1. 解題說明:

   依題目要求，將敘述以if-else形式呈現。

2. 校能分析:

   - S(P)=4*(n+1), 4 words(m, n, 回傳值, 回傳位址), n+1次遞迴
   - T(P)=2*(n+1), 2 steps(if-else, return), n+1次遞迴
   - f(n)=O(n)

3. 測試與驗證

   測試: cout<<recA(1, 1);// ←'3'

   驗證: recA(1, 1)=recA(1-1, recA(1, 0))=recA(0, recA(0, 1))=recA(0, 2)=3

nonrecursive function:

```
int nonrecA(int m, int n){//以stack實作蝶帶版本
    int stackOfM[100]={0}, current=0;
    stackOfM[0] = m;
    while(current>=0){
        m = stackOfM[current--];
        if(m==0) n++;
        else if(n==0){
            n=1;
            stackOfM[++current]=m-1;
        }else{
            stackOfM[++current]=m-1;
            stackOfM[++current]=m;
            n--;
        }
    }
    return n;
}
```

1. 解題說明:

    以stack替代遞迴，其中if-else中做的判斷及行為都等 同於recA中的if-else式。

2. 效能分析:

    - S(P)=4+0, 4 words(m, n, stackOfM, current)
    - 4+5*(n+1)≤T(P)≤4+6*(n+1), 4 for {宣告*2, while, return},
    - f(n)=O(n)

3. 測試與驗證:

    測試: cout<<nonrecA(1, 1);// ←'3'

    驗證: 分析nonrecA(1, 1):

    (1) m=1, n=1, stackOfM={1, 0, …}, current=0

    (2) m=1, n=0, stackOfM={1-1, 1, …}, current=1

    (3) m=0, n=1, stackOfM={0, 1-1, …}, current=1

    (4) m=0, n=2, stackOfM={0, 0, …}, current=0

    (5) m=0, n=3, stackOfM={0, 0, …}, current=-1

    最後回傳n，即3。

problem 2

Problem 2:

If S is a set of n elements, the *powerset* of S is the set of all possible subsets of S. For example, if $S = (a,b,c)$, then *powerset* $(S) = \{( ), (a), (b), (c), (a,b), (a,c), (b,c), (a,b,c)\}$. Write a recursive function to compute *powerset* (S).

完整程式碼:

```
/*  1017 hw2 資工二乙 41143264 楊育哲
    遞迴產生排列 ex. S=[a, b] => powerset(S)={' ', a, b, ab}
*/
#include <iostream>
#include <string>
using namespace std;

class Powerset{
private:
    string power_set[100];//最多可能有2^26個組合, 此處預設測資組合數低於一百
    char S[26], comb[27];//26個字母+'\0'
    int top, end, Sindex;//top協助comb成為stack, end為資料集大小, Sindex協助power_set存資料
public:
    Powerset(char *s, int size){//初始化, s為資料集, size為資料集長度
        top = 0;
        Sindex = 0;
        end = size;
        copy(s, s+size, S);
    }
    void rec_PS(int start){//遞迴主程式, 運作方式與課本舉例之排列遞迴類似
        for(int i=0; i<start; i++) power_set[Sindex]+=comb[i];
        Sindex++;
        for(int i=start; i<end; i++){
            comb[top++] = S[i];
            rec_PS(i+1);
            comb[--top]='\0';
        }
    }
    int strPriority(string str){//計算字串依字典的優先度, 協助排列函式運作
        int count=0;
        for(int h=0; h<str.length(); h++){
            if(str[h]!='\0') count=count*26+str[h]-'a'+1;
        }
        return count;
    }
    void setSort(){//依字典順序將power_set排列
        for(int i=0; i<Sindex-1; i++){
            int index=i;
            for(int j=i+1; j<Sindex; j++){
```

```
                if(strPriority(power_set[index])>strPriority(power_set[j])) index=j;
            }
            string tmp=power_set[index];
            power_set[index] = power_set[i];
            power_set[i] = tmp;
        }
    }
    void outputSet(){//輸出，輸出前先作排列
        setSort();
        for(int i=0; i<Sindex; i++) cout<<power_set[i]<<"\n";
    }
};

int main(){
    char A[3]={'a', 'b', 'c'};
    Powerset test(A, 3);
    test.rec_PS(0);
    test.outputSet();
    return 0;
}
```

recursive function:

```
    void rec_PS(int start){//遞迴主程式, 運作方式與課本舉例之排列遞迴類似
        for(int i=0; i<start; i++) power_set[Sindex]+=comb[i];
        Sindex++;
        for(int i=start; i<end; i++){
            comb[top++] = S[i];
            rec_PS(i+1);
            comb[--top]='\0';
        }
    }
```

1. 解題說明:

   與課本中排列遞迴的函式類似。comb用來記錄組合，每次遞迴就將comb存進所求的power_set中。此處的for迴圈可以以深度優先的形式尋訪樹，產生所要的所有組合，也因此並不會有排序的狀況，所以得在加寫setSort()函式。

2. 效能分析:

   - S(P)=5*(2^n), 5words(變數∗5), 2^n次遞迴
   - T(P)=4*(n+1)*n/2+3*2^n, 兩個for（4*(n+1)）+其他的遞迴（3*2^n）
   - f(n)=O(n^2)

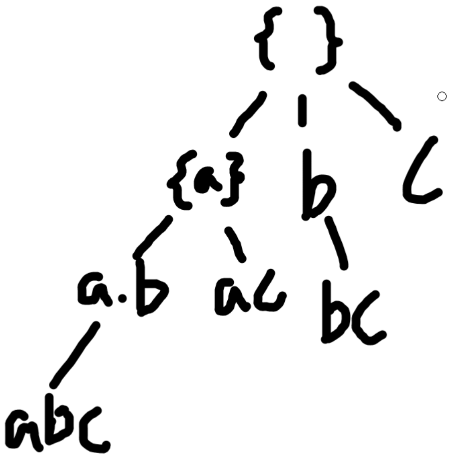
1. 測試與驗證

   測試: 結果:

```
int main(){
    char A[3]={'a', 'b', 'c'};
    Powerset test(A, 3);
    test.rec_PS(0);
    test.outputSet();
    return 0;
}
```

a
b
c
ab
ac
bc
abc

驗證:



依深度優先‘ ‘, a, ab, abc, ac, b, bc, c依序加進power_set

排列後為{‘ ‘, a, b, c, ab, ac, bc, abc}