CH1 Time Complexity

- recurisve
  - Factorial
  - Fibonacci Number
    - (F_n =

      $$\{ 0, \quad \text{if } n = 0 \backslash [6pt] 1, \quad \text{if } n = 1 \backslash [6pt] F_{n-1} + F_{n-2}, \quad \text{if } n \geq 2$$

)
    - the number of recursive calls grows exponentially with n is $1.41^n < F_n < 2^n$
    - use DP skill need $O(n)$
  - Binomial Coefficient
    - (\binom{n}{m} =

      $$\{ 1, \quad \text{if } n = m \text{ or } m = 0 \backslash [6pt] \binom{n-1}{m} + \binom{n-1}{m-1}, \quad \text{otherwise}$$

)
    - use DP skill need $O(nk)$
  - GCD
    - (\mathrm{GCD}(A, B) =

      $$\{ A, \quad \text{if } A \bmod B = 0 \backslash [6pt] \mathrm{GCD}(B, A \bmod B), \quad \text{otherwise}$$

)
  - Ackerman function
    - (A(m, n) =

      $$\{ n + 1, \quad \text{if } m = 0 \backslash [6pt] A(m-1, 1), \quad \text{if } n = 0 \backslash [6pt] A(m-1, A(m, n-1)), \quad \text{otherwise}$$

)
  - Tower of Hanoi $O(2^n)$
    - (T(n) =

      $$\{ 1, \quad \text{if } n = 1 \backslash [6pt] 2T(n-1) + 1, \quad \text{if } n \geq 2$$

)
  - recursive
    - Factorial
    - Fibonacci Number
      - F_n =
        - 0, if n = 0

                ■ 1, if n = 1
                ■ F_{n-1} + F_{n-2}, if n ≥ 2
            ■ the number of recursive calls grows exponentially with n: 1.41^n < F_n < 2^n
            ■ use DP skill need O(n)
        ■ Binomial Coefficient
            ■ C(n, m) =
                ■ 1, if n = m or m = 0
                ■ C(n-1, m) + C(n-1, m-1), otherwise
            ■ use DP skill need O(nk)
        ■ GCD
            ■ GCD(A, B) =
                ■ A, if A mod B = 0
                ■ GCD(B, A mod B), otherwise
        ■ Ackerman function
            ■ A(m, n) =
                ■ n + 1, if m = 0
                ■ A(m - 1, 1), if n = 0
                ■ A(m - 1, A(m, n - 1)), otherwise
        ■ Tower of Hanoi O(2^n)
            ■ T(n) =
                ■ 1, if n = 1
                ■ 2T(n-1) + 1, if n ≥ 2

- permutation: $O(n! * n)$

```
void swap(char *a, char *b){
  char temp=*a;
  *a=*b;
  *b=temp;
}
void perm(char *list, int i, int n){
    int j, temp;
    if(i==n){
      for(j=0;j<n;j++)
        printf("%c", list[j]);
      printf("\n");
    }
    else{
      for(j=i;j<n;j++){
        swap(&list[i], &list[j]);//list[j]當head
        perm(list, i+1, n);//後面(i+1)~n permutation
        swap(&list[i], &list[j]);//還原
      }
    }
}
int main(){
    char list[3]={"abc"};
    perm(list,0,3);
    return 0;
}
output:
abc
```

```
acb
bac
bca
cba
cab
```

- basic math
    - (\sum_{i=1}^{n} i^2=\frac{n(n+1)(2n+1)}{6})
    - (\sum_{i=1}^{n} i^d \approx n^{d+1}, d\ge0)
    - (\sum_{i=1}^{n} \frac{1}{i} = \log n)
    - (n! \le n^n)
        - (\lg(n!) = \Theta(n \log n))
    - (\frac {n}{2} ^ \frac {n}{2} \le n!)
    - Stirling's Formula
        - (n! = \sqrt{2 \pi n} \left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right) \approx n^{,(n + \frac{1}{2})} \times e^{-n})
    - ((\log n)^b = o(n^a), \quad a > 0)
        - e.g. ((\log n)^{100} < n^{0.0001})
        - e.g. $(\log n)^{100} < n^{0.0001}$
    - (\log^*(\log n) = \log*^ n - 1)

- Master Theorem
    - $T(n) = aT(\frac{n}{b}) + f(n)$

- extended Master Theorem
    - $T(n) = aT(\frac{n}{b}) + nlgn$

CH2 CH4 Array & Linked Lisd

CH3 Stack & Queue

- stack
    - stack application
        - parsing context-free languages
        - evaluating arithmetic expressions(infix, postfix, prefix)
        - function call management
        - recursion removal/recursive call
        - traversing tree(preorder, inorder, postorder)
        - DFS graph traversal
        - eight queen problem
        - maze problem
        - reverse output
        - 客人取盤子行為

- stack implementation
    - array
    - linked list
    - two queues
- stack permutations
    - $\frac{1}{n+1}\binom{2n}{n}$
    - 與下列問題同義
        - the number of binary tree structures with n nodes
        - the number of valid parentheses wih n "("and")"
        - the number of matrix multiply chain with n+1 matrix($\because$ 有n個*)
        - the number of train output order with n trains in the gateway
- Infix to Postfix

```
InfixtoPostfix(Infix){
  while(Infix has not been scanned over){
    x=NextToken(Infix);
    if(x is operand)//x是operand
      print(x);
    else{//x是operator
      if(x==')'){
        while(stack.top()!='('){
          y=stack.top();
          stack.pop();
          print(y);
        }
      }
      else{
        if(precedence(x)>precedence(stack.top()))
          stack.push(x);
        else{
          while(precedence(x)<=precedence(stack.top())){
            y=stack.top();
            stack.pop();
            print(y);
          }
          stack.push(x);
        }
      }
    }
  }
  while(!stack.empty()){//清空stack
    y=stack.top();
    stack.pop();
    print(y);
  }
}
```

- Postfix求值

```
Evaluate(Postfix){
  whlie(Postfix has not been scanned over){
    x=NextToken(Postfix);
    if(x is oeprand)
      stack.push(x);
    else{//x is operator
      right_operand=stack.pop();
      left_operand=stack.pop();
      stack.push(left_operand opeartor right_operand);//依operator作運算，
放入stack
    }
  }
  result=stack.top();
  stack.pop();
  return result;
}
```

- check for balanced brackets(){}[]

```
bool judge(s:string){
  while(s has not been scanned over){
    x=NextToken(s);
    if(x=='('||x=='['||x=='{')
      stack.push(x);
    else{
      if(stack.isempty())
        return false;
      else{
        if(x==')'){
          if(stack.top()!='(')
            return false;
        }
        if(x==']'){
          if(stack.top()!='[')
            return false;
        }
        if(x=='}'){
          if(stack.top()!='{')
            return false;
        }
        stack.pop();
      }
    }
  }
  if(stack.isempty())
    return true;
  return false;
}
```

- queue
  - queue implementaion

        - circular array with no tag -> n-1
        - circular array with tag -> n
        - single linked list
        - circular linked list
        - two stacks

CH5 Tree & Binary Tree

- Tree
    - ancestor=predecessor
    - descendent=successor
    - tree化成binary tree, binary tree化成tree
        - tree化成binary
            - Leftmost-child-Next-Right-sibling
    - Forest化成binary tree, binary tree化成Forest
        - 皆針對Root做操作

- Binary Tree
    - ith level max node=2^{i-1}
    - height h max node=$2^h - 1$
    - leaf num=n_0,degree-2=n_2, $n_0 = n_2 + 1$
    - 不可決定唯一binary tree
        1. preorder+postorder
        2. level-order+preorder
        3. level-order+postorder
        4. BST+inorder
    - the number of different binary trees with n nodes
        - Catalan number
            - $\frac{1}{n+1}\binom{2n}{n}$

- Binary Search Tree
    - In a BST find i-th smallest data

```
struct Node {
  Node* Lchild;
  int data;
  int Lsize;
  Node* Rchild;
};

search(T:BST, i:int){//在T中找出i-th小之data
  if(T!=Nil){
    k=(T->Lsize)+1;//代表root是kth小的data
    if(i==k)
      return T->Data;
    else if(i<k)
      return serach(T->Lchild,i);//去左子樹找i-th小
    else
```

```
        return search(T->Rchild,i-k);//去右子樹找(i-k)th小
    }
  }
```

- Heap
  - build a heap with n nodes
    - Top-Down
      - $O(nlogn)$
    - Bottom-Up
      - $O(n)$
  - Heapify[adjust(tree,i,n)]

```
void adjust(int tree[], int i, int n){
  //調整以i node no.為root之子樹成為Heap
  int j=2*i;//目前j是i之左子點No.
  int x=tree[i];
  while(j<=n){//尚有兒子
    if(j<n && tree[j]<tree[j+1])
      j=j+1;
    if(x>=tree[j])
      break;
    else{
      tree[j/2]=tree[j];//上移至父點
      j=2*j;//新的左子點位置
    }
  }
  tree[j/2]=x;//x置入正確格子中
}
void buildheap(int tree[], int n){
  for(int i=n/2;i>=1;i--)
    adjust(tree, i, n);
}
```

- Disjoin Sets
  - Union
  - Find
- Thread Binary Tree

CH9 Advanced Tree

- Double-Ended Priority Queue
  - Min-Max Heap
  - Deap
  - SMMH
- Extended Binary Tree

    - E=I+2N
    - Huffman Algorithm
- AVL Tree
- M-way search tree
    - B Tree of order m
    - B^+ Tree of order m
- Red-Blcak tree
- Optimal Binay Search Tree(OBST)
- Splay Tree
- Leftist Heap
- Binomail Heap
- Fibonacci Heap

CH7 Sort

- Search

    - Linear Search
    - Binary Search

- Sort

    - Elementary/Simple Sorts
        - Insertion sort
        - Selection sort
        - Bubble sort
        - Shell sort
    - Advanced/Efficient Sorts
        - Quick sort
        - Merge sort
        - Heap sort
    - Linear-Time sorting methods
        - LSD Radix sort=Radix sort
        - MSD Radix sort=Bucket sort
        - Counting sort

CH8 Hashing

- Collision

- Overflow

- Identifier Density

- Loading Density

- Hashing 優點

- hashing function design

- 3 disgn criteria
    - 計算簡單
    - 碰撞少
        - perfect hashing function
    - 不要造成hash table局部偏重儲存的情形
        - uniform hashing function
- 常見hashing function design methods
    - Middle Square
    - Mod(Division)
    - Folding Addition
    - Digits Analysis

- Overflow Handling

    - Linear Probing
    - Quadratic Probing
    - Double Hashing
    - Chaining
    - Rehashing

CH6 Graph

- DFS

    - adjacency matrix: $O(V^2)$
    - adjacency lists: $O(V+E)$

- BFS

    - adjacency matrix: $O(V^2)$
    - adjacency lists: $O(V+E)$

- Topological sort

    - adjacency lists: $O(V+E)$

- Minimum Spanning Tree

    - Kruskal's algorithm
        - adjacency matrix: $O(ElogE)$
        - adjacency lists : $O(ElogE)$
            - compare to prim's: $\because E << V^2 \therefore logE = O(logV), \therefore O(ElogV)$
    - Prim's algorithm
        - adjacency matrix: $O(V^2)$
        - binary heap+adjacency lists: $O(ElogV)$
        - Fibonacci heap+adjacency lists: $O(E+VlogV)$
    - Sollin's algorithm

- Shortest Path Length

    - single source to other destinaitons
        - Directed Acyclic Graph(DAG)

        - adjacency lists: $O(V+E)$
    - Dijkstra algorithm
        - adjacency matrix: $O(V^2)$
        - binary heap+adjacency lists: $O(ElogV)$
        - Fibonacci heap+adjacency lists: $O(E+VlogV)$
    - Bellman-Ford Algorithm
        - adjacency matrix: $O(V^3)$
        - adjacency lists: $O(VE)$
  - all pairs of vertex
    - Floyd-Warshall algorithm
        - adjacency matrix: $O(V^3)$
    - Johnson's algorithm
        - adjacency matrix: $O(V^2logV+VE)$

- AOE network

- Articulation Point

- Biconnected Graph

  - a connected undirected graph with no AP

- Biconnected component

  - G' is a subgraph of G, and G' is a biconnected graph
  - G' is Maximum Component