Data Structure

Midterm Cover: Algorithm, some basic data structure including array, stack, queue, linked list.

Algorithm Analysis

If we assume that computer execute one line basic code need one computation, then we can use n to represent the complexity of the program.

for example, **for** loop need (n+1) execution, and **print** need one execution.

Then, Therefore, we have diffrent notation to roughly judge an algorithm, and We call that **asymptotic notation**

Asymptotic Notation

- [ ] Big-O notation represents the upper bound of the running time of an algorithm. Thus, it gives the **worst-case** complexity of an algorithm.

```
O(g(n)) = { f(n): there exist positive constants c and n0
            such that 0 ≤ f(n) ≤ cg(n) for all n ≥ n0 }
```

*represent the **UPPER** bound*

- [ ] However, **Omega notation** represents the lower bound of the running time of an algorithm. Thus, it provides the **best case** complexity of an algorithm.

```
Ω(g(n)) = { f(n): there exist positive constants c and n0
            such that 0 ≤ cg(n) ≤ f(n) for all n ≥ n0 }
```

- [ ] **Theta notation** encloses the function from above and below. Since it represents the upper and the lower bound of the running time of an algorithm, it is used for analyzing the **average-case** complexity of an algorithm.

- if a function `f(n)` lies anywhere in between `c1g(n)` and `c2g(n)` for all `n ≥ n0`, then `f(n)` is said to be asymptotically tight bound.
- Notice:

```
显然如果 T(n) = n^2，那么 T(n) = O(n^2)，T(n) = O(n^3)，T(n) = O(n^4) 都是成立的，但是因为第一个 f(n) 的增长速度与 T(n) 是最接近的，所以第一个是最好的选择，所以我们说这个算法的复杂度是 O(n^2)

对于顺序执行的语句或者算法，总的时间复杂度等于其中最大的时间复杂度。max(O(n^2), O(n))，即 O(n^2)

对于条件判断语句，总的时间复杂度等于其中 时间复杂度最大的路径 的时间复杂度
```

Example:

```java
long aFunc(int n) {
    if (n <= 1) {
        return 1;
    } else {
        return aFunc(n - 1) + aFunc(n - 2);
    }
}
// T(n) = T(n - 1) + T(n - 2); The time complexity is O (2^N)
// The space complexity is O (N)
```

Sorting

Insertion Sort

```
INSERTION-SORT(A)
   for i = 1 to n
    key ← A [i]
      j ← i – 1
     while j > = 0 and A[j] > key
      A[j+1] ← A[j]
      j ← j – 1
    End while
    A[j+1] ← key
  End for

  // everytime choose the smallest one in the left.
  // Time complexity: O(N^2) and best case is O(N),
```

```
  // Space complexity is O(1)
```

- NOTE:

  one for loop, and one while loop to choose the smallest each time. and we dont need extra space for it.

Merge Sort

```
ERGE(A,p,q,r)
    n1=q-p+1;
    n2=r-q;
    create new arrays L[n1+1] and R[n2+1]
    for i=0 to n1-1
        L[i]=A[p+i]
    for j=0 to n2-1
        R[j]=A[q+1+j]
    L[n1]=1000//假设1000是无穷大
    R[n2]=1000
    i=j=0
    for k=p to r
        if(L[i]<=R[j])
            A[k]=L[i]
            i=i+1
        else
            A[k]=R[j]
            j=j+1
MERGE_SORT(A,p,r)
    if p<r
        q=floor((p+r)/2)
        MERGE_SORT(A,p,q)
        MERGE_SORT(A,q+1,r)
        MERGE(A,p,q,r)
```

Quick Sort

```
quicksort(A,p,r)

if p < r;

   then q<- partition(A,p,r)

            quicksort(A,p,q-1)

            quicksort(A,p+1,r)

其中

partition(A,p,r)

x<-A[r]

i<-p-1

for j<-p to r-1
```

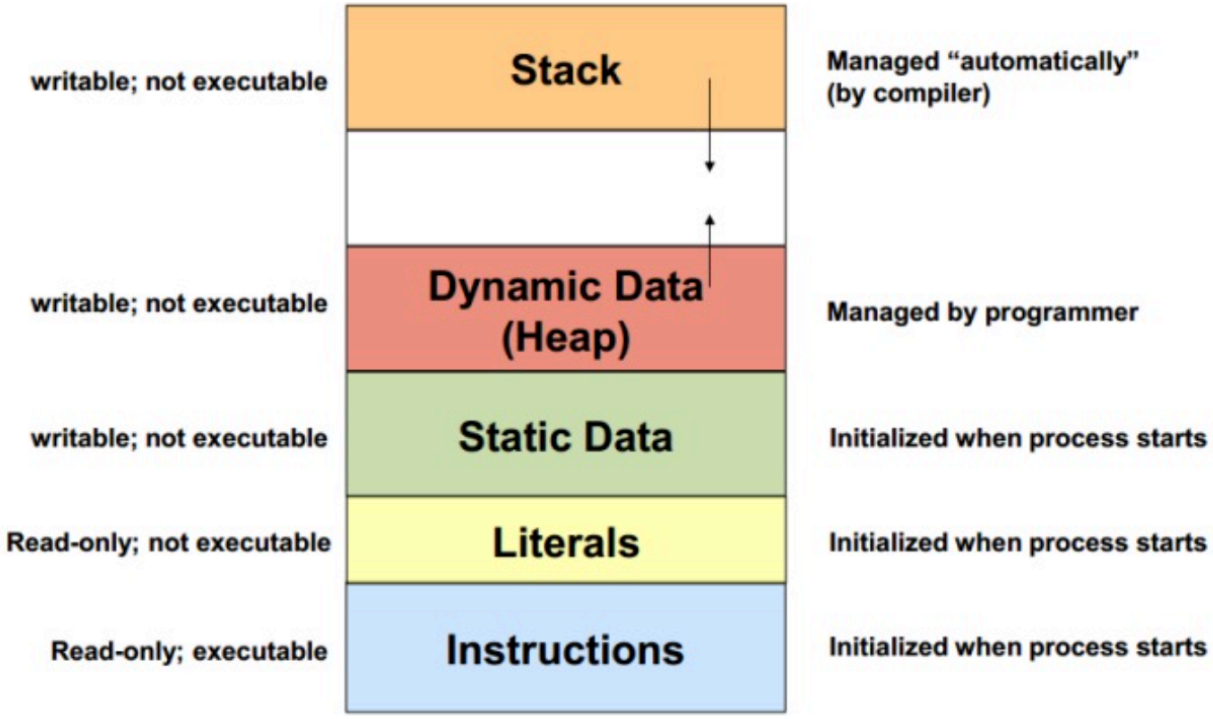
```
    do if A[j]<= x
            then i++;
                    exchange A[i]<->A[j]
exchange A[i+1]<->A[r]
return i+1
```

Basic Data Structure

Array

memory allocation

How computer allocate the memory: Think it as a large unti, which contains stack heap, stactic data, literals instructions.



Problem && Answer:

Static全局变量与普通的全局变量有什么区别?
全局变量（外部变量）的说明之前再冠以static就构成了静态的全局变量。全局变量本身就是静态存储方式，静态全局变量当然也是静态存储方式。

这两者在存储方式上并无不同。这两者的区别在于非静态全局变量的作用域是整个源程序，当一个源程序由多个原文件组成时，非静态的全局变量在各个源文件中都是有效的。而静态全局变量则限制了其作用域，即只在定义该变量的源文件内有效，在同一源程序的其它源文件中不能使用它。由于静态全局变量的作用域限于一个源文件内，只能为该源文件内的函数公用，因此可以避免在其他源文件中引起错误。

static全局变量与普通的全局变量的区别是static全局变量只初始化一次，防止在其他文件单元被引用。

static函数与普通函数有什么区别?

static函数与普通的函数作用域不同。尽在本文件中。只在当前源文件中使用的函数应该说明为内部函数（static），内部函数应该在当前源文件中说明和定义。对于可在当前源文件以外使用的函数应该在一个头文件中说明，要使用这些函数的源文件要包含这个头文件。
****static函数与普通函数最主要区别是static函数在内存中只有一份，普通静态函数在每个被调用中维持一份拷贝程序的局部变量存在于（堆栈）中，全局变量存在于（静态区）中，动态申请数据存在于（堆）**

Java中：

static表示“全局”或者“静态”的意思，用来修饰成员变量和成员方法，也可以形成静态static代码块，但是Java语言中没有全局变量的概念。
用public修饰的static成员变量和成员方法本质是全局变量和全局方法，当声明它类的对象市，不生成static变量的副本，而是类的所有实例共享同一个static变量
只要这个类被加载，Java虚拟机就能根据类名在运行时数据区的方法区内找到他们。因此，static对象可以在它的任何对象创建之前访问，无需引用任何对象。

The difference between Stack and Heap:

Basic difference is that stack memory is allocated in a contiguous block, and heap is allocated in any random order. and heap is memory fragmentation.

Primitive type versus objects: **When you pass an argument of a primitive type to a method, Java copies the value of the argument into the parameter variable. As a result, changes to the parameter variable have no effect on the argument.**

Initialization

need to define the lenghth of the array when initialize it

```
private static final int Num = 5;
double[] scores = new double[Num];

// note that "final" means once you decare it, it can not change, can not assign new value to it.
// when use "final" to define a class, it means this class wont have children class.
```

Passsing arrays as parameters

```
swapElements(array, i, n – i – 1)
```

why java pass the value of array but not address(reference)?

Answer:

```
java has basic rule for passing the parameter

A method cannot modify a parameter of primitive type (that is, numbers or Boolean values).
A method can change the state of an object parameter.
A method cannot make an object parameter refer to a new object.

which means:
Java defines all arrays as objects, implying that the elements of an array are shared between
the callee and the caller.
**which means array element is not an object**

But why we dont need pass the reference? because java has encapsulation for pointer and
refernce.
```

Insertion

```
How to implement the insertion in array

1. check whether the array is full or index out of range
2. create the new array which length is greater than original one
3. find the position of index where you want to insert, and keep the element before index the
same
4. insert the element move the element after the index
```

Deletion

```
1. check the array is empty or index out of range
2. keep the element before index the same and remove the index element
3. move the element after index one foward
```

Checking indendity

```
1. check the lenghth first
2. check every element one by one
3. if matched, return true, else false.
```

Return a new array

```
public class returnArrayDemo
 public static void main(String arg[])
  char[] c;
  c = vowels();
  for(int i = 0; i < c.length; i++)
    System.out.println(c[i]);

public static char[] vowels()
char[] newArray = new char[5];
newArray[0] = 'a';
newArray[1] = 'e';
newArray[2] = 'i';
```

```
newArray[3] = 'o';
newArray[4] = 'u';
return newArray;
```

Sorting an array

- selection sort
- quick sort
- bubble sort

List

what is linked List: Consists of a series of nodes (Node class)

- each linked list contains a data and a reference
- not neceessarily adjacent in memory
- flexible on element insertion and deletion

why add a *head* in linked list?

head is a node that **does not store the data**

Initialization

```
public class Node(){
  public T nodeValue;
  public Node<T> next;

  public Node(){
    nodeValue = null;
    next = null;
  }

  public Node(T item){
    nodeValue = item;
    next = null;
  }

  public Node(T item, Node nextnode){
    nodeValue = item;
    next = nextnode;
  }
}
```

```
Node<String> p = new Node<String>("red");
Node<String> q = new Node<String>("green")

p.next = q;       // link p to q         p --> q

Node<String> front = p;    // set front to point at the first node: front --> p --> q
```

Find insertion position

```
Node<T> curr = front; // start at the front of list
for (int i = 0; i < xPos; i++)
  curr = curr.next;// move along list
```

Notice:

T是**自定义泛型**，泛型的主要目的是实现 java的类型安全，消除了强制类型转换

Insertion

To insert a new node before a node referenced by 'curr', the code must have access to the previous node, 'prev', since its link must be changed.

```
two step
1. newnode.next = curr
2. prev.next = newnode
```

The insertion is O(1) since only two links need to be changed. But the real cost is the sequential search to find the insertion position, which is O(n).

Deletion

```
// find the deletion position
for (int i = 0; i < xpos; i++)
  curr = curr.next;
// connect prev to curr.next
prev.next = curr.next;
curr.next = null;
```

Notice: if the node need to be removed is the first one, then we need to set the front = curr.next, and curr.next = null

However, for singly linked list, we need to have to pointer start from front and scan, since we need to find two node: curr and prev.

```
 /* Delete the first occurrence of the target in the linked list referenced by front; return
the value of front */
{
   // initialize pointers
   Node<T> curr = front;
   Node<T> prev = null;
   boolean foundItem = false;
  while (curr != null && !foundItem)
   if (target.equals(curr.nodeValue)) {
        if (prev == null)// remove first Node (O(1)) front = front.next;
              front = front.next;
         else       // erase middle Node (O(1))
              prev.next = curr.next;
         curr.next = null;
         foundItem = true;
```

```
   else   // advance curr and prev
         prev = curr;
         curr = curr.next;

 return front;  // may be updated
}  // end of remove()
```

Question:

What will happen if there is a 'head' for implementing remove()?
How to swap two elements (at position i and j) in a linked list?

Answer2:

1、确定两个节点的先后顺序
2、next、prev互相交换顺序以及将换向前方的节点与之前的节点对接。（1.prev.next = 2）
3、判断是否相邻

Stack

Using linked list to implement the stack

```
1. create a calss named Node, which include the initialization, and getelement()
2. create a class stack
3. implement the function include pop(), push(), peak(), isempty(), is full()..
The parameter in stack should include maxsize, currentsize, Node header(This is important, it
should be the top element in the stack)
```

Using array to implement the stack

```
class ArrayStack{
    private String[] data=new String[10];//存储数据
    private int size;//记录个数

    public void push(String str) {
      //判断是否需要扩容
      if (size>data.length) {
        data=Arrays.copyOf(data, data.length*2);
      }
      data[size++]=str;
    }

    public String peek() {
    //判断栈是否为空
      if (size==0) {
        throw new EmptyStackException();
      }
      return data[size-1]; //获取栈顶元素
    }
```

```
    public String pop() {
      String str=this.peek();//获取栈顶元素
      size--;                 //减少元素个数
      return str;
    }

    public boolean empty() {
      return size==0;
    }
    //返回对象在堆栈中的位置，以 1 为基数
    public int search(String str) {

      //顺着放倒着拿（FILO/LIFO）
for (int i = size-1; i >=0; i--)
{
        if (str==data[i]||str!=null&&data[i].equals(str)) {
          return size-i;          }
      }
      return -1;    //返回栈中不存在该元素
    }
  }

}
```

Queue

Using linked list to implement the queue

```
public class Queue_List(){
  int currentsize;
  int maxsize = MAX;
  Node head = new Node();;
  Node rear;
}

public class makeEmpty(){
  Node temp = head;
  Node follow = head;
  while (temp != rear){
    follw = temp;
    temp = temp.next;
    follow = null;
  }
  currentsize == 0;
}

public class boolean IsFull(){
  return currentsize == maxsize;
}

public class boolean IsEmpty(){
  return currentsize == 0;
```

```
}

public class boolean Enqueue_Node(Object item){
  if (IsFull()){
    return false
  }
  else if (IsEmpty){
    currentsize++;
    Node newnode = new Node(item);
    head.next = newnode;
    rear = newnode;
  }
  else{
    currentsize++;
    Node newnode = new Node(item);
    newnode.next = head.next;
    head.next = newnode;
  }
  return true;
}

public class Node Dequeue_Node(){
  if (IsEmpty()){
    return null;
  }
  else{
    currentsize--;
    Node temp = head;
    while (temp.next != rear) temp = temp.next;
    rear = temp;
    temp = temp.next;
    return temp;
  }
}
```

Notice: The garbage collection of java in linked list

The garbage collector periodically checks for unreachable objects, such as your deleted node here (assuming there are no other references.)

Precisely when it does this is subject to a lot of conditions, since there are multiple garbage collector implementations that use different timing strategies. In general you cannot predict when.

The garbage collector periodically checks for unreachable objects, such as your deleted node here (assuming there are no other references.)

Precisely when it does this is subject to a lot of conditions, since there are multiple garbage collector implementations that use different timing strategies. In general you cannot predict when.

If you need to eagerly release some resource on delete(for example, perhaps your linked list node has an associated file descriptor), you are better off doing it yourself before dropping the last reference.

Using array to implement the queue

there are two options to implement an queue by using the array.

option 1:

◦ Enqueue at data[0] and shift all of the rest of the items in the array down to make room.

◦ Dequeue from data[numItems-1]

option 2:

◦ Enqueue at data[rear+1]
◦ Dequeue at data[front]
◦ The rear variable always contains the index of thelast item in the queue.
◦ The front variable always contains the index of the first item in the queue.
◦ When we reach the end of the array, wrap around to the front again.

option 2 implementation

```
public class Queue_Array {
  int capacity;
  int front, rear, size;
  ElementType[] array;
}

// create an empty queue
public void makeEmpty() {
  size = 0;
  front = 1; rear = 0;
  array = new Object[capacity];
}

public boolean isFull(){
  return size == capcity;
}

public boolean isEmpty(){
  return size == 0;
}

public boolean enqueue(Object x){
  if (isfull()){
    return false ;
  }
  else {
    size++;
    rear = (rear+1)%capacity;
    array[rear] = x;
    return true;
  }
}

public <T> dequeue(){
  int first;
  if (isEmpty()){
```

```
    return null;
  }
  else{
    size--;
    first = front;
    front = (front +1)%capacity;
    return array[first];
  }
}
```

How about option 1?

basically the same as option 2, the difference is that we need to use the varirable numofitem and variable front to get the index of enqueue and dequeue element. May need more calculation to maintain the array order.