作业1.

1. public class AssignmentOfTheory {  
 public static void main(String[] args) {  
 for (int i = 1; i < 100; i+=4) {  
 System.*out*.println(*func1*(i)>*func2*(i));  
 }  
 }  
 private static int func1(int n){  
 return 100\*n\*n;  
 }  
 private static int func2(int n){  
 return (int) Math.*pow*(2,n);  
 }  
}

前四个是true，即1，5，9，13是true，改变程序为：

public class AssignmentOfTheory {  
 public static void main(String[] args) {  
 for (int i = 1; i <= 15; i++) {  
 System.*out*.println(*func1*(i)>*func2*(i));  
 }  
 }  
 private static int func1(int n){  
 return 100\*n\*n;  
 }  
 private static int func2(int n){  
 return (int) Math.*pow*(2,n);  
 }  
}

发现14是true，15是false，因此是n<=14时后一个比前一个短。

作业2：

1.

Z=x+y是7500次

Z=x-y是2500/4=625,我用程序模拟如下：

private static void ass2(){  
 int n = 10000;  
 int a = 0;  
 int b = 0;  
 while(n>=0){  
 n--;  
 if(Math.*random*()<=0.75){  
 a++;  
 } else if (Math.*random*()<=0.25) {  
 b++;  
 }  
 }  
 System.*out*.println(a+","+b);  
}

结果为：

7483,662

7510,647

7568,609

7380,648

7508,637

7502,611

7521,606

7528,655

7502,589

7454,666

2. (1)n次

2. (2)n(n+1)(n+2)/6

作业3:

1.A最小，因为logn<n<n^i(i>0)

2.

private static int ass3\_2(int[] arr){  
 int i = 0;  
 int j = arr.length-1;  
 while(i<j){  
 if(i==arr[i]){  
 return i;  
 } else if (i<arr[i]) {  
 i = arr[i];  
 } else {  
 i++;  
 }  
 }  
 return -1;  
}

作业4：

1.使用快速幂算法计算实数的整数幂即可

用自然语言解释就是寻找离幂次数最近的2的整数倍，然后2的整数倍这一部分可以二分，

其他的部分乘上去就行。需要注意的是二分并不一定对所有情况都是最佳的，但确实是一个最通用且较为快速的办法。

public class FastPow {  
 public static double fastPow(double x,int n){  
 double ans = 1;  
 long y = n;  
 if(n<0){  
 y=-y;  
 x=1/x;  
 }  
 while(y>0){  
 if(y%2==1){  
 ans\*=x;  
 }  
 x=x\*x;  
 y/=2;  
 }  
 return ans;  
 }  
  
 public static void main(String[] args) {  
 System.*out*.println(*fastPow*(7,65));  
 }  
}

2.

（1）线性表

（2）树

（3）集合（这里的集合是指HashSet）

（4）（6）线性表

（5）我不太懂用什么好，但一定不是线性表

作业5：

1.尾部的插入删除操作要提供，头部的可以不提供 。

2.loop（遍历）,contains（是否包含某元素(返回布尔值)）,remove,insert

作业8：

1. BCA,CAB

2.(1)√(2)×，只需要移动top指针

3.这句话正确，但是并不好，原因是增加了代码量负担的同时并没有显著地让时间空间复杂度变好。但需要指明的是，这种思想是有实际作用的，这种做法在纯函数式编程语言中也是有用的。（原题见力扣232）

作业9:

1.

中缀：A+B\*C-D/E

后缀：ABCDE/-\*+

2.

前缀：-\*+ABC/DE

中缀：A+B\*C-D/E

作业12：

1.先序序列是ABCDEFG，中序序列不可能是:C

(二叉树的先序序列和中序序列是进栈出栈的关系)

2.前序遍历：根左右，中序遍历：左根右，后序遍历：左右根

（1）把根左右和左根右的左字删掉，都成了根右，因此是任何结点的左子树全为空

（2）右子树空掉

（3）只有唯一结点的二叉树

（4）层数小于等于2 的二叉树或只有左子树的二叉树

3.

4.B

5.

**两种建堆算法：威廉姆斯建堆算法和弗洛伊德建堆算法  
\* 1.威廉姆斯的建堆算法分为两步：  
\* 1.1和父元素比，如果比父元素小就放在父元素尾部  
\* 1.2如果比父元素大，要把父元素向下调整，再与更上层父元素比较，直到比父元素小或者到达最顶部  
\* 时间复杂度为O(n\*logn)  
\* 2.弗洛伊德的建堆算法也分为两步：  
\* 2.1找到最后一个非叶子节点  
\* 2.2从后向前，对每个结点执行<下潜>  
\* 时间复杂度为O(n)  
\* <下潜>是指把一个结点和它的两个子节点中较大的进行交换,然后看有没有更多的左右孩子，如果有，继续下潜**

威廉姆斯建堆算法：

public int[] heapify(int[] arr){  
 for (int i = 1; i < arr.length; i++) {  
 int parent = arr[(i-1)>>1];  
 int child = arr[i];  
 int p = i;  
 int q = (i-1)>>1;  
 while(child>parent){  
 swap(p,q);  
 p = q;  
 q = (p-1)>>1;  
 parent = arr[q];  
 child = arr[p];  
 }  
 }  
   
}

弗洛伊德建堆算法：

public MaxHeap(int[] arr){   
 this.array = arr;  
 this.size = arr.length;  
 heapify();  
}

public void heapify(){  
 for (int i = (size>>1)-1; i >= 0 ; i--) {   
 down(i);  
 }  
}

//下潜

private void down(int parent) {  
 int left = (parent<<1)+1;  
 int right = left + 1;  
 int max = parent;//先假定父亲的更大  
 if(left<size&&array[left]>array[max]){  
 max = left;  
 }  
 if(right<size&&array[right]>array[max]){  
 max = right;  
 }  
 if(max!=parent){//找到了更大的孩子  
 swap(parent, max);  
 down(max);  
 }  
}

作业13：

1. n-1

2. n

3. 21

4.



