以下代码的执行结果是().

```
1   int main(){
2         int i=-2147483648;
3         return printf("%d,%d,%d,%d",~i,-i,1-i,-1-i);
4   }
```

正确答案: D 你的答案: 空 (错误)

```
0,2147483648,2147483649,2147483647
0,-2147483648,-2147483647,2147483647
2147483647,2147483648,2147483649,2147483647
2147483647,-2147483648,-2147483647,2147483647
```

表达式 a+b*c-(d+e)/f 的后缀表达式为()

正确答案: C 你的答案: 空 (错误)

```
abc*+def+/-
+*-/+bcaefd
abc*+de+f/-
abc*+de+f-/
```

四分位数是统计学的一个概念，把序列中的数值由小到大排列并分成四等分，处于三个分割点位置的数就是四分位数。n 为序列的总长度，三个四分位数可以根据如下公式求出：
Q1 的位置=（n+1）x 0.25
Q2 的位置=（n+1）x 0.5
Q3 的位置=（n+1）x 0.75
比如数据序列：1,3,5,7,2,4,6
由小到大排列的结果是：1,2,3,4,5,6,7
一共 7 项，Q1 的位置=(7+1)*0.25=2,Q2 的位置=(7+1)*0.5=4，Q3 的位置=(7+1)*0.75=6，四分位数即为第 2,4,6 个元素上对应的数值：（2,4,6）
那么数据序列 6,45,49,16,42,41,7,38,43,40,36 的四分位数为：

正确答案: B 你的答案: 空 (错误)

```
49,41,43
16,40,43
16,45,7
16,41,45
```

一棵深度为 5 的完全二叉树最少有()个节点。(第一层深度视为 1）

正确答案: B 你的答案: 空 (错误)

```
15
16
31
```

32

在有序表(5,8,36,48,50,58,88)中二分查找字 58 时所需进行的关键字比较次数是（），对应的判定树高度为（）.

正确答案: B　你的答案: 空 (错误)

2,2
2,3
3,2
3,3

假设以行优先顺序存储三维数组 A[5][6][7],其中元素 A[0][0][0]的地址为 1100，且每个元素占 2 个存储单元，则 A[4][3][2]的地址是()

正确答案: D　你的答案: 空 (错误)

1150
1291
1380
1482

马路上有编号 1,2,3...10 的十盏路灯，为节约用电而又不影响照明，可以把其中 3 盏灯关掉，但不可以同时关掉相邻的两盏，在两端的灯都不能关掉的情况下，有()种不同的关灯方法。

正确答案: A　你的答案: 空 (错误)

20
60
120
240

房间里有 8 人，分别佩戴着从 1 号到 8 号的纪念章，任选 3 人记录其纪念章号码，最大的号码为 6 的概率()

正确答案: B　你的答案: 空 (错误)

3/28
5/28
23/28
25/28

58 同城北京租房列表页共有 3 个广告位，广告库中共有 5 个经纪人，每个经纪人发布了2 条广告房源参与此列表页3 个广告位的随机展示(即每条广告房源获得展示的概率是一样的),则此列表页展示时，同时展示同一个经纪人的两条房源的概率是（）

正确答案: A　你的答案: 空 (错误)

1/3
2/9
7/27
3/50

定义 bash 环境的用户文件是？

正确答案: C 你的答案: 空 (错误)

bash &.bashrc
bash & bash_profile
bashrc &.bash_profile
bashrc &.bash_conf

数组 A 由 1000W 个随机正整数(int)组成，设计算法，给定整数 n，在 A 中找出符合如下等式:n=a+b 的 a 和 b，说明算法思路以及时间复杂度是多少?

参考答案

将数组排序，杂度 n*logn 在从头开始,假设第 i 个位置时 arr[i],那就在 i 到 1000 万之间找 n - arr[i] 二分查找的效率是logn,由于当arr[i] > n/2 时就不用找了,所以最终效率 2*n*logn

数据库中有学院表和成绩表
学院表 t_school 结构如下:
学院 ID：school_id,学院名称:school_name
成绩表 t_score 结构如下:
学号:id.姓名:name,分数:score,学院 ID:school_id
请用 sql 语句查询出学院名称为"计算机系"的分数最高的前 20 位的学生姓名

参考答案

select score.name from t_school school, t_score score where school.school_id = score.school_id and school.school_name="计算机系" order by score.score desc limit 20

斗地主是中国非常流行的一种牌类游戏:一副扑克 54 张牌，3 人轮抓，每人 17 张，3 张底牌。请问，同一个人 17 张手牌就抓到火箭(即同时抓到大小王)的概率是多少？说明计算过程

参考答案

先求必须任一一个大小王不在底牌，即：C（52,3)/C(54,3)，这保证了大小王一定在上面牌中

然后一个人抓到第一张王的概率=17/51,抓到第二张王概率=16/50，所以单独一个人抓到概率为:17/51*16/50，那么 3 人中出现一个人的概率是 3*17/51*16/50
所以最后结果为:C（52,3)/C(54,3)*3*17/51*16/50=0.3081

请用时间复杂度最低的方法找出数组中数值差距最大的两个元素的差值？

```
1    public static int findTheNumber(int[] array) {
2            if(null == array || 0 == array.length) return 0;
3            int min = array[0], max = array[0];
4            for(int i = 1; i < array.length; i++) {
5                    if(array[i] > max) {
6                            max = array[i];
7                    }
8                    if(array[i] < min) {
9                            min = array[i];
10                   }
11           }
12           return max - min;
13   }
```

实现一个优先级消息队列。
不妨将消息抽象成一个整数，该整数数值代表消息的优先级。优先级消息队列是一个这样的队列：任何时间都有可能有消息入队列，任何时间都有可能消息出队列。但只能弹出当前保存的优先级最高的消息。

```
1    class CPriorityMsgQueue
2    {
3        public:
4                        //任意消息进入队列
5                    void enQueue(int msg);
6                    //优先级最高的消息弹出队列
7                    int deQueue();
8        private:
9                     //可以自行添加需要的私有成员
10   }
```

1.消息队列

消息队列是项目中经常需要使用的模式，下面使用 STL 的 queue 容器实现：

[cpp] view plaincopy

```
1. #include <queue>     // STL header file for queue
```

```
2.  #include <list>
3.  using namespace std;  // Specify that we are using the std namespace 4.
5. class Message; 6.
7. class Message_Queue 8.
{
9.    typedef queue<Message *, list<Message *> > MsgQueType;
10.
11.   MsgQueType m_msgQueue;
12.
13. public :
14.   void Add(Message *pMsg)
15.   {
16.     // Insert the element at the end of the queue
17.     m_msgQueue.push(pMsg);
18.   }
19.
20.   Message *Remove()
21.   {
22.     Message *pMsg = NULL;
23.
```

```
24.      // Check if the message queue is not empty
25.      if  (!m_msgQueue.empty())
26.      {
27.        // Queue is not empty so get a pointer to the
28.        // first message in the queue
29.        pMsg = m_msgQueue.front();
30.
31.        // Now remove the pointer from the message queue
32.        m_msgQueue.pop();
33.      }
34.      return  pMsg;
35.   }
36.
37.   int GetLength() const
38.   {
39.      return  m_msgQueue.size();
40.   }
41. };
```

2.优先级消息队列

上面的消息队列类只支持在队列的尾部添加一个元素，在许多的应用程序中，需要根据消息的优先级将消息添加到队列中，当一个高优先级的消息来了，需要将

该消息添加到所有比它优先级低的消息前面，下面将使用 priority_queue 来实现优先级消息队列类。

函数对象(Functors)

优先级消息队列的实现和上面消息队列实现相似，唯一不同的是这里使用了函数对象来决定优先权 CompareMessages，该结构重载了操作符"(,)"，该机制可以实现可以将一个函数作为一个参数，和函数指针对比有如下好处：

1. 函数对象效率更高，可以是内联函数，函数指针总是有函数调用的开销。
2. 函数对象提供了一个安全的实现方法，这样的实现不会出现空指针访问。

[cpp] view plaincopy

```
1.  #include <queue>      // STL header file for queue
2.  #include <list>
3.  using namespace  std;  // Specify that we are using the std namespace 4.
5. class Message; 6.
7. class Priority_Message_Queue
8. {
9.      struct  Entry
10.     {
11.         Message *pMsg;
12.     int priority; 13.
        };
14.
15.     struct  Compare_Messages
16.     {
```

```
17.       bool operator () ( const Entry& left , const Entry& right)
18.       {
19.           return (left.priority < right.priority);
20.       }
21.    };
22.
23.   typedef priority_queue<Entry, vector<Entry>, Compare_Messages  >
24.           Message_Queue_Type;
25.
26.   Message_Queue_Type m_message_Queue;
27.
28. public :
29.
30.   void Add(Message *pMsg, int  priority)
31.   {
32.      // Make an entry
33.      Entry entry;
34.      entry.pMsg = pMsg;
35.      entry.priority = priority;
36.      // Insert the element according to its priority
37.      m_message_Queue.push(entry);
38.   }
```

```
39.
40.  Message *Remove()
41.  {
42.    Message *pMsg = NULL;
43.
44.    // Check if the message queue is not empty
45.    if  (!m_message_Queue.empty())
46.    {
47.      // Queue is not empty so get a pointer to the
48.      // first message in the queue
49.      pMsg = (m_message_Queue.top()).pMsg;
50.
51.      // Now remove the pointer from the message queue
52.      m_message_Queue.pop();
53.    }
54.    return  (pMsg);
55.  }
56.
57.  size_t Get_Length() const
58.  {
59.    return  m_message_Queue.size();
60.  }
```

```
61. };
62.
```