

## راه حل سوال های کانتست (Div2) UICPC Round #23

# سوال A: <mark>1454A</mark>

- 🍫 هر جایگشتی با خاصیت موجود برای این سوال چاپ کنیم کافی خواهد بود.
- با توجه به اینکه n>=2 میباشد، حتما چنین جایگشتی وجود خواهد داشت و در واقع برای هر nای که در نظر بگیریم، به تعداد جابجایی ها جواب وجود خواهد داشت که کد های زیر یکی از این جواب هارا تولید میکنند:

## C++:

```
#include <bits/stdc++.h>
using namespace std;
typedef long long 11;
int main(){
    11 t;
    cin >> t;
    while(t--){
        11 n;
        cin >> n;
        for(ll i=n;i>=1;i--){
            if(n\%2 \&\& i==n/2+1){
                 cout<<i-1<< ' '<<i<< ' ';
                 i--;
                 continue;
            cout<<i<' ';
        }
        cout<<endl;</pre>
    }
    return 0;
```

#### Python:

```
input()
a=sorted(map(int,input().split()))[::-1]
a[:2]=a[1::-1]
if a[1]<a[0]+a[2]:print('YES',*a)
else:print('NO')</pre>
```



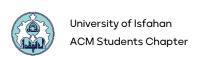
- تنها کافیست برای یک ز، چک کنیم اعداد موجود در خانه های 1-j و 1+j در شرایط موجود صدق میکنند یا نه و یعنی فقط
  سه تایی های متوالی را بررسی کنیم. اگر هیچ سه تایی در شرایط صدق نکند، یعنی آرایه تا جایی نزولی است و از آنجا به
  بعد کاملا صعودیست که باعث میشود هیچ جوابی نداشته باشیم.(پیچیدگی زمانی؟)
- راه با پیچیدگی زمانی n^2 هم آن است که برای هر ز چک کنیم آیا عددی کوچکتر در قبل آن و سپس در بعد آن وجود دارد
   یا نه؟( آیا میتوانید با استفاده از ذخیره کردن اطلاعات این راه را به به راهی با (o(n) تبدیل کنید؟)

#### C++:

```
#include <bits/stdc++.h>
using namespace std;
typedef long long 11;
int main(){
    11 t;
    cin >> t;
    while(t--){
        11 n; cin >> n;
        std::vector<11> p(n);
        for(ll i=0;i<n;i++)cin >> p[i];
        bool ok = false;
        for(ll j=1;j<n-1;j++){</pre>
             if(p[j]>p[j-1]&&p[j]>p[j+1]){
                 ok = true;
                 cout<<"YES"<<endl;</pre>
                 cout<<j-1+1<<' '<<j+1<<' '<<j+1+1<<endl;</pre>
                 break;
             }
        }
        if(!ok)cout<<"NO"<<endl;</pre>
    }
    return 0;
}
```

# Python Python

```
for _ in range(int(input())):
    n = int(input())
    a = List(map(int, input().split()))
    for i in range(1,n-1):
        if a[i-1]<a[i]>a[i+1]:
            print("YES",i,i+1,i+2)
            break
    else:print("NO")
```



## سوال C: <mark>1515B</mark>

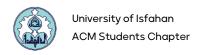
- اگر n را بتوان به شکل 2x یا 4x نوشت که x یک عدد مربع است، پاسخ YES است. در غیر این صورت NO
   است.
- برای تجسم این ساختار، ابتدا با ساختن یک مربع کوچکتر با استفاده از ۲ یا ۴ قطعه شروع می کنیم. فقط
   کافی است از x تا از آن مربع های کوچکتر برای ساختن یک مربع بزرگتر استفاده کنیم.
- بیایید ثابت کنیم که هیچ پاسخ دیگری وجود ندارد . بیایید هر قطعه مثلث را طوری تعریف کنیم که یک ضلع کوتاه به طول ا و یک ضلع بلندتر به طول رادیکال ۲ داشته باشد. یک ضلع مربع را در نظر بگیرید، و فرض کنید که ۵ مثلث در ضلع کوتاه و b مثلث در ضلع بلندتر دارد. طول ضلع طلع معلی خواهد بود. مساحت مربع یک عدد گویا است زیرا مساحت هر قطعه مثلث گویا است. بنابراین، ۵(a+2-√b) باید گویا باشد، به این معنی که ۵ ما است، یا 0 است. اگر هر کدام ه باشد، می توانیم از ساختار پاراگراف قبل استفاده کنیم.

#### C++:

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
bool sq(int n){
    int a = sqrt(n);
    return a*a==n;
}
int main(){
    LL t;
    cin >> t;
    while(t--){
        LL n;
        cin >> n;
        if(n%2==0&&sq(n/2))cout<<"YES"<<endl;
        else if(n\%4==0\&\&sq(n/4))cout<<"YES"<<endl;
        else cout<<"NO"<<endl;</pre>
    }
    return 0;
}
```

# Python:

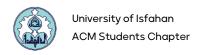
```
for i in range(int(input())):
    n=int(input())
    print('YES' if n%2==0 and n//2==round((n//2)**0.5)**2 or n%4==0 and
n//4==round((n//4)**0.5)**2 else 'NO')
```



- ما به رشته های واقعی اهمیت نمی دهیم، تمام اطلاعاتی که نیاز داریم این است تعداد جفت ها {ه,ه}, {۱,ه}, {ه,ا},
   {۱٫۱} آن را بشماریم و سپس الگوریتمی حریصانه را برای اولین بازیکن دنبال میکنی: سعی کنید ایندکس با {۱٫۱} در صورت وجود بیابیم، و بعد {۱٫۵}، و بعد {۱٫۵} و در نهایت {۱٫۵}.
  - 💠 برای بازیکن دوم استراتژی مشابه: اول {۱،۱}، از {۱،۵}، از {ه،۱}، از {ه،۵}.
    - 💠 پس از آن فقط مقایسه کنید که چه کسی ا بیشتر دارد.

#### C++:

```
#include <bits/stdc++.h>
using namespace std;
typedef long long 11;
int main(){
    11 n;
    string s1;
    string s2;
    cin >> n >> s1 >> s2;
    11 f=0, c=0, s=0;
    for(ll i=0;i<2*n;i++){</pre>
         if(s1[i]=='1'&&s2[i]=='1')
             C++;
         else if(s1[i]=='1')
             f++;
         else if(s2[i]=='1')
             S++;
    }
    f = c\%2 ? f+1:f;
    if(s>f+1){
         cout<<"Second"<<endl;</pre>
    }
    else if(f>s){
         cout<<"First"<<endl;</pre>
    }
    else{
         cout<<"Draw"<<endl;</pre>
    }
    return 0;
}
```



#### Python:

```
n = int(input())
a, b = input(), input()
t = {i + j: 0 for i in '01' for j in '01'}
for i in range(2 * n): t[a[i] + b[i]] += 1
d = t['11'] & 1
d += (t['10'] - t['01'] + 1 - d) // 2
if d > 0: d = 1
elif d < 0: d = 2
print(['Draw', 'First', 'Second'][d])</pre>
```

وکتور α را اینگونه در نظر میگیریم که [0] اولین زیر دنباله ی صعودی است [۱] دومین زیر دنباله صعودی است و به همین ترتیب، سپس با باینری سرچ، هریک از اعضای آرایه را به زیر دنباله ی مناسب آن اضافه میکنیم. میدانیم برای [i] num زیر آرایهی صعودی مناسب، زیر آرایه ای با کمترین اندیس ممکن است که عدد آخر آن، از [i] num کوچکتر باشد. پس بر همین اساس باینری سرچ را می نویسیم.

## C++:

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll mx = 2*1e5;
vector<\(\lambda \lambda \);</pre>
vector<LL> a[mx];
int main(){
    LL n;
    cin >> n;
    for(ll i=0;i<n;i++){</pre>
         cin >> num[i];
    for(ll i=0;i<n;i++){</pre>
         LL 1=-1, h=n, mid=n;
         while(l<h-1){</pre>
             mid = (1+h)/2;
             if(a[mid].size()==0||a[mid][a[mid].size()-1]<num[i]){</pre>
                  h=mid;
              }
             else{
                  l=mid;
         }
         a[h].push_back(num[i]);
    for(ll i=0;i<n;i++){</pre>
         for(ll j=0;j<a[i].size();j++){</pre>
             cout<<a[i][j]<<' ';
             if(j==a[i].size()-1)cout<<endl;</pre>
         }
    }
    return 0;
}
```

