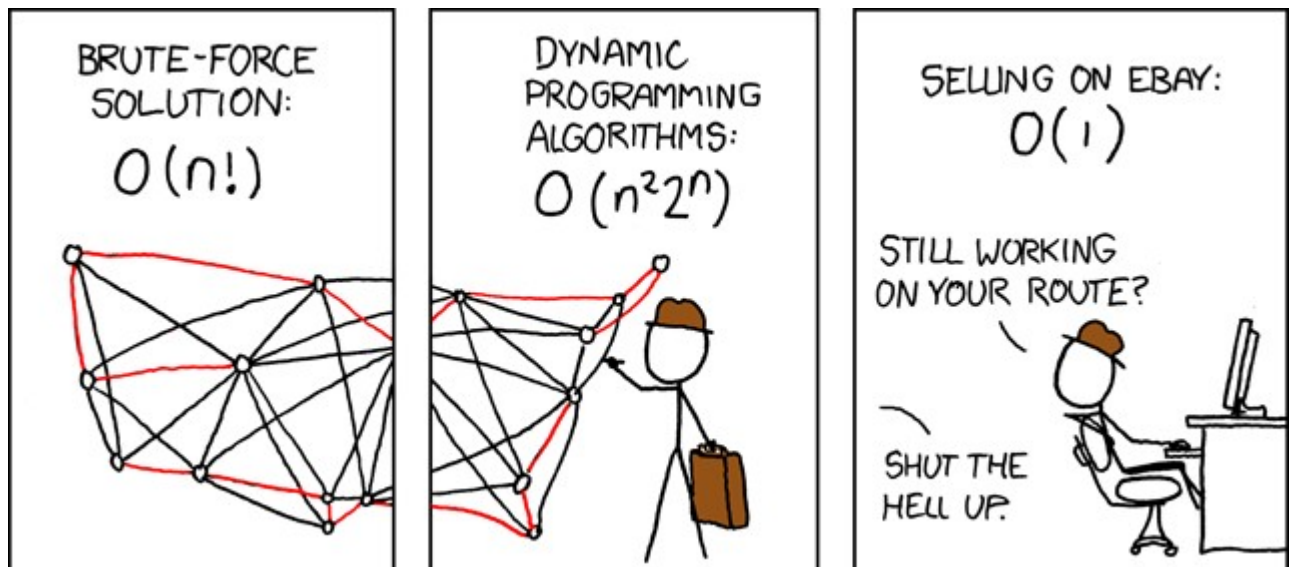# Competitive Programming Hakpak

by Evan Partidas

xkcd: Travelling Salesman Problem

```
public static int BS(int[] arr, int query){
     int low = 0,high = arr.length;
     while(low<high){
          int mid = (low+high)/2;
          //Insert comparator here
          int cmp = arr[mid] - query;
          if(cmp==0){
               return mid;
          }
          else if(cmp>0){
               high = mid;
          }
          else {
               low = mid+1;
          }

     }
     return -1;
}
```

Regular Binary search.

-Search Complexity of O(log n) where n is the size of the array.

-Will return an index of the number `query`.

-Will return -1 if the element is not present.

-Array must be sorted.

-Can be modified to work for any comparator simply by modifying the way `cmp` is calculated.

```
public static int BFS(int[] arr, int query){
     //Binary Floor Search
     int low = -1,high = arr.length;
     while(low+1<high){
          int mid = (low+high)/2;
          //Insert comparator here
          int cmp = arr[mid] − query;
          if(cmp>0){
               high = mid;
          }
          else {
               low = mid;
          }
     }
     return low;
}
```

Floored Binary search.

-Search Complexity of O(log n) where n is the size of the array.

-Will return the largest index that contains a value less than or equal to the given `query.`

-Will return -1 if no element that satisfies the conditions is present.

-Array must be sorted.

-Can be modified to work for any comparator simply by modifying the way `cmp` is calculated.

-Useful when searching for up to what point a given input/solution/etc. is valid.

For c++ look into function upper_bound and lower_bound in the algorithm library. upper_bound can be used as a floor search by subtracting 1 from the **MEMORY LOCATION** returned by it.

Segmented Tree

```java
//All global variables needed
int lo[],hi[],delta[],query[],size;

//This is use-defined. (Note sums don't work with this implementation of lazy
propagation)

        static final int IDENTITY = Integer.MAX_VALUE;//Identity for Min

        int queryFunction(int a,int b){
                return Math.min(a,b);//Min Function (Clearly)
        }
```

```java
public SegmentTree(int n){
        size = 4*n;
        lo = new int[size];
        hi = new int[size];
        delta = new int[size];
        query = new int[size];

        init(0,0,n-1);
}
```

```java
void init(int node, int li,int ri){

        lo[node] = li;
        hi[node] = ri;
        if(li==ri)
                return;
        int mid = (li+ri)/2;
        init(node*2+1,li,mid);
        init(node*2+2,mid+1,ri);
}
```

```java
void prop(int node){
        delta[2*node+1]+=delta[node];
        delta[2*node+2]+=delta[node];
        delta[node]=0;
}
```

```java
void update(int node){
        query[node] = queryFunction(
                query[2*node+1]+delta[2*node+1],
                query[2*node+2]+delta[2*node+2]);
}
```

```java
int query(int node,int li,int ri){
        if(ri<lo[node]||hi[node]<li)
                return IDENTITY;

        if(li<=lo[node]&&hi[node]<=ri)
                return query[node]+delta[node];

        prop(node);

        int ret = queryFunction(
                query(2*node+1,li,ri),
                query(2*node+2,li,ri));
        update(node);

        return ret;

}
```

```java
void increment(int node,int li,int ri,int val){
                if(ri<lo[node]||hi[node]<li)
                        return;

                if(li<=lo[node]&&hi[node]<=ri){
                        delta[node]+=val;
                        return;
                }

                prop(node);

                increment(2*node+1,li,ri,val);
                increment(2*node+2,li,ri,val);

                update(node);

}
```

Suffix Array

```c
int N,C[MAXN],RA[MAXN],TMP[MAXN],SA[MAXN],LCP[MAXN];
char STR[MAXN];
```

```c
void countsort(int k,int size){
//Stable Countsort
  int i;
  for(i=0;i<size;i++)
        C[i]=0;
  for(i=0;i<N;i++)
    C[i+k<N?RA[i+k]:0]++;
  for(i=1;i<size;i++)
    C[i]+=C[i-1];
  for(i=N-1;i>=0;i--)
    TMP[--C[SA[i]+k<N?RA[SA[i]+k]:0]] = SA[i];
  for(i=0;i<N;i++)
    SA[i]=TMP[i];
}
```

```c
int lcp(){
    int len=0;
    int prefix=0;
    int i;
    for(i=1;i<N;i++)
        TMP[SA[i]]=SA[i-1];
    for(i = 0;i<N-1;i++){
        while(STR[i+len]==STR[TMP[i]+len]) len++;
        prefix = max(len,prefix);
        len = max(len-1,0);
    }
    return prefix;
}
```

```c
void initSA(){
    int i,k=1,r;
    for(i=0;i<N;i++)
        RA[i]=STR[i],SA[i]=i;
    for(k=1;k<N;k<<=1){
        countsort(k,max(N+1,300));
        countsort(0,max(N+1,300));
        TMP[SA[0]]=r=0;
        for(i=1;i<N;i++){
            TMP[SA[i]]=
                (RA[SA[i]]==RA[SA[i-1]] && RA[SA[i]+k]==RA[SA[i-1]+k])?r:++r;
        }
        for (i = 0; i < N; i++)
            RA[i] = TMP[i];
        if (RA[SA[N-1]] == N-1) break;
    }
}
```