TO SILVER ALGORITHMS CHEAT SHEET

Min, Max O(n), O(1) if sorted

```
//given int[] x:
//find position of max
int maxId = -1; //will store index of max
for(int i=0;i<x.length;i++) {
  if(maxId==-1 || x[i]>x[maxId]) maxId = i;
  //use < sign for minimization
  //use compareTo() w/ objects (beware null)
}
//If sorted just get first/last item
```

Searching for item in array O(n)

```
//given int[] x; //not sorted
int foundIndex = -1;
for(int i=0;i<x.length;i++) if(x[i]==y) {
  foundIndex = i; //we're looking for y here
}
```

Binary Search O(log(n))

```
//given int[] x; //sorted
for(int low=0,high=x.length-1;low<=high;) {
  int mid = low+(high-low)/2;
  if(y < x[mid]) high = mid – 1;
  else if(y > x[mid]) low = mid + 1;
  else {
    foundIndex = mid;
    break; //we found y at mid
  }
}
```

Find Duplicates O(n*n),n if sorted

```
//given int[] x; //not sorted
int dupId = -1;
for(int i=0;i<x.length;i++)
  for(int j=0;j<x.length;j++)
    if(x[i]==x[j]) { //use compareTo w/ obj
      dupId = i;
      break;
    }
//given int[] y; //sorted
int dupId = -1;
for(int i=0;i<x.length-1;i++)
  if(x[i]==x[i+1]) {
    dupId = i;
    break;
  }
```

Finding pair sums to S, O(n) sorted

```
//use double for loop if not sorted
//if x is sorted:
int[] y = copyOf(x); //you need to write
for(int i=0;i<y.length;i++) y[i] = s-y[i];
boolean success = false;
for(int i=0,j=y.length-1;i<j;) {
  if(x[i]==y[j]) { success = true; break; }
  else if(x[i]>y[j]) j--;
  else i++;
}
```

Java Arrays/Collections Sort

```
Arrays.sort(x); //sorts x reference
//SEE JAVA CHEAT SHEET FOR COLLECTIONS.SORT
```

Mergesort

```
public static void mergesort(int[] x) {
  if(x.length <= 1) return;
  int q = x.length/2, n=x.length;
  int[] a=Arrays.copyOfRange(x,0,q);
  int[] b=Arrays.copyOfRange(x,q,x.length);
  mergesort(a);
  mergesort(b);
  merge(x,a,b);
}
static void merge(int[]x,int[]a, int[]b) {
  for(int i=0,j=0,k=0;k<x.length;k++) {
    if(j==b.length||a[i]<b[j]) x[k]=a[i++];
    else x[k]=b[j++];
  }
}
```

QuickSort

```
public static void sort(int[]x,int i,int j){
  int index = partition(x,i,j);
  if(i < index – 1) sort(x,i,index-1);
  if(index < j) sort(x,index,j);
}
public static void part(int[]x,int i,int j){
  for(int pivot = x[(i+j)/2];i <= j) {
    while(x[i]<pivot) i++;
    while(x[i]>pivot) j--;
    if(i <= j) {
      int tmp = x[i];
      x[i++] = x[j];
      x[j--] = tmp;
    }
  }
}
```

Base Conversion to Decimal

```
String chars = "0123456789ABCDEF";
String x = "101010101";
int base = 2, ex = 1, out = 0;
for(int i=x.length()-1;x>=0;x--) {
  out+=ex*chars.indexOf(x.charAt(i)+"");
  ex*=base;
}
//use BigInteger for larger numbers
```

Base Conversion from Decimal

```
String chars = "0123456789ABCDEF";
String out = ""; //out may end up long
int base = 2, x = 12345;
if(x==0) out = "0";
while(x>0) {
  out=chars.charAt(x%base)+out;
  x/=base;
}
//use shift operators (>>, <<) to multiply
or divide binary numbers
//For example in binary x*17 is x*16+x
//Which is x shifted left 2
```

Permutations:

```
//Each recursive call removes an item
void setup() { perm("ABC","",3); }
void perm(String x, String pre, int len) {
  if(len==0) { println(pre); return; }
  for(int i=0;i<x.length();i++) {
    String p = pre+x.charAt(i);
    String c =
x.substring(0,i)+x.substring(i+1);
    perm(c,p,len-1);
  }
}
//outputs ABC,ACB,BAC,BCA,CAB,CBA
```

Variations:

```
void setup() { combos("ABC","",2);  }
void combos(String x, String pre, int len) {
  if(len==0) { println(pre); return; }
  for(int i=0;i<x.length();i++) {
    String p = pre+x.charAt(i);
    combos(x,p,len-1);
  }
} //note recursive does not remove items
//outputs AA,AB,AC,BA,BB,BC,CA,CB,CC
```

Combinations:

```
void setup() {  //main if not processing
  comb("ABC",new boolean[3],0);
} //use bool array to say include/or not
void comb(String x, boolean[] inc, int pos){
  if(pos==inc.length) {
    for(int i=0;i<inc.length;i++) {
      if(inc[i]) print(x.charAt(i));
    }
    println(); return;
  }
  inc[pos] = true; comb(x,inc,pos+1);
  inc[pos] = false;comb(x,inc,pos+1);
} //outputs ABC,AB,AC,A,BC,B,C
```

Combo of fixed length:

```
void setup() {
  cb("ABC", new boolean[3], 0, 2);
} //checks if len matches # of true in y
void cb(String x,boolean[]y,int i,int len){
  int sumtrue = 0;
  for (int j=0;j<y.length;j++) {
    sumtrue += y[j]?1:0;
  }
  if(i==y.length && sumtrue==len) {
    for(int j=0;j<y.length;j++) {
      if(y[j]) print(x.charAt(j));
    }
    println(); return;
  }
  if(sumtrue>len || i==y.length) return;
  y[i] = true;  cb(x, y, i+1,len);
  y[i] = false; cb(x, y, i+1,len);
}
//OUTPUTS: AB,AC,BC
```

Flood Fill Count Adjacent:

```
void setup() {
  char[][] x = {{'#',' ','#'},
                {' ',' ','#'},
                {'#','#','#'}};
  println(count(x,2,2));
  //note that you need to remove \0
  //characters (or change them back)

  //another option is to make a copy
  //of the map before operating

  //a third option is to create a
  //boolean[][] visited
}

int count(char[][]x,int r,int c) {
  char visitedchar = '\0'; //special char
  char lookingfor = '#';
  if(r<0||r>=x.length||c<0||c>=x[r].length)
return 0;
  int out = 0;
  if(x[r][c]==visitedchar) return 0;
  if(x[r][c]==lookingfor) out++;
  else return 0;
  x[r][c] = visitedchar;
  int[][] d={{-1,1,0,0},{0,0,-1,1}};
  for(int i=0;i<4;i++)
    out+=count(x,r+d[0][i],c+d[1][i]);
  return out;
}
```

Distance on 2d grid w/ flood fill:

```
void setup() {
  char[][] x = {{'#','#','#'},
                {' ',' ','#'},
                {'X','#','#'}};
  println(dfs(x,0,0,'X'));
  //same comments as previous example
  //with regards to \0 chars or visited[][]
}

int dfs(char[][]x,int r,int c,char y) {
  char visitedchar = '\0'; //special char
  char lookingfor = '#';
  if(r<0||r>=x.length||c<0||c>=x[r].length)
return -1;
  if(x[r][c]==visitedchar) return -1;
  if(x[r][c]==y) return 0;
  else if(x[r][c]!=lookingfor) return -1;
  x[r][c] = visitedchar;
  int[][] d={{-1,1,0,0},{0,0,-1,1}};
  for(int i=0;i<4;i++) {
    int t=dfs(x,r+d[0][i],c+d[1][i],y);
    if(t>=0) return 1+t;
  }
  return -1;
}
```

Dijkstra:

```
class N implements Comparable<N> {
  int id;
  Map<N,Integer>e=new HashMap<N,Integer>();
  Integer d;
  N previous; //in path
  N(int id) { this.id = id; }
  int compareTo(N o) {
    if (this.d==null&&o.d==null) return 0;
    else if (this.d==null) return 1;
    else if (o.d==null) return -1;
    else return d.compareTo(o.d);
  }
  void addNeighbor(N node, int d) {
    e.put(node, d);
  }
}
Integer dijkstra(ArrayList<N> x,N a, N b){
  for (N n : x) n.d = null;
  a.d = 0;
  PriorityQueue<N>pq=new PriorityQueue<N>();
  pq.add(a);
  while (pq.size ()>0) {
    N cur = pq.poll();
    for (N n : cur.e.keySet()) {
      int newD=cur.d+cur.e.get(n);
      if (n.d==null || newD < n.d) {
        n.d = newD;
        n.previous = cur;
        pq.remove(n);
        pq.add(n);
      }
    }
    if (cur==b) return cur.d;
  }
  return null;
}
```

Graph diameter given start:

```
int getLongestDistanceTo(N a) {
 //set all Ns to have dist of null
 PriorityQueue<N>pq=new PriorityQueue<N>();
 pq.add(a);
 a.d = 0;
 Integer max = null;
 while(pq.size()>0) {
  N cur = pq.poll();
  for(N n : cur.e.keySet()) {
   int newD = cur.d+cur.e.get(n);
   if(n.d == null || newD < n.d){
    n.d = newD;
    pq.remove(n);
    pq.add(n);
   }
  }
  if(max==null||cur.d>max) max=cur.d;
 }
 return max;
}
```

Generic 2D Dynamic Programming:

```
class TwoDimDynamicProgramming<I,J,V> {
 public Map<I, Map<J, V>> d;
 public TwoDimDynamicProgramming() {
  d=new LinkedHashMap<I, Map<J,V> >();
 }
 public boolean solved(I i, J j) {
  if(!d.containsKey(i)) return false;
  if(!d.get(i).containsKey(j))return false;
  return true;
 }
 public V solution(I i, J j) {
  return d.get(i).get(j);
 }
 public void addSolution(I i, J j, V v) {
  if(!d.containsKey(i))
   d.put(i, new LinkedHashMap<J,V>());
  d.get(i).put(j,v);
 }
}
```

Dynamic Programming Example:

```
//http://www.usaco.org/index.php?page=viewpr
oblem2&cpid=107
class BaleShare extends USACOProblemTester{
 public static void main(String[] args) {
  new BaleShare();  }
 public BaleShare() {
super("http://www.usaco.org/current/data/bal
eshare.zip"); }
 public void solve() throws IOException {
  int MX = 100*(20/3+1);
  int n = nextInt(), sum = 0;
  dp = new Boolean[n][MX][MX];
  int[] s = new int[n];
  for(int i=0;i<n;i++)sum+=(s[i]=nextInt());
  int answer = MX;
  for(int i=0;i<MX;i++)
   for(int j=0;j<MX;j++){
    if(check(s,n-1,i,j))
     answer=Math.min(answer,Math.max(
      i,Math.max(j,sum-(i+j))));
   }
  println(answer);
 }
 Boolean[][][] dp;
 boolean check(int[]s,int n,int i,int j) {
  if(n==0 && i==0 && j==0) return true;
  if(n<0 || i<0 || j<0) return false;
  if(i > j) { int tmp = i; i = j; j = tmp; }
  if(dp[n][i][j]!=null) return dp[n][i][j];
  boolean answer = false;
  if(check(s,n-1,i,j)) answer = true;
  else if(check(s,n-1,i-s[n],j))answer=true;
  else if(check(s,n-1,i,j-s[n]))answer=true;
  dp[n][i][j] = answer;
  return answer;
 }
}
```