Indukcí dokažte následující výrok: pokud $left$ a $right$ jsou parametry funkce f a platí $left \leq right$, pak volání $f(array, left, right)$ vrátí minimální hodnotu z hodnot všech prvků v poli $array$ na indexech $left$ až $right$ včetně a vždy skončí v konečném počtu kroků.

```
int f(int[] array, int left, int right) {
  if (left == right) return array[left];
  int l = f(array, left, (left + right) / 2);
                      // celociselne deleni, zaokrouhluje dolu
  int r = f(array, (left + right) / 2 + 1, right);
  return (l < r) ? l : r;
}
```

Dlouhou vodorovnou čarou oddělte důkaz prvního a druhého indukčního kroku, u druhého indukčního kroku zřetelně napište, jak vypadá indukční předpoklad a co se z něj snažíte dokázat.

První krok: máme dokázat, že pro $left = right$ vrátí volání $f(array, left, right)$ po konečném počtu kroků minimum z množiny $array[left]$. To zjevně platí: funkce se vrátí hned na prvním řádku (na kterém nejsou žádné cykly ani volání dalších funkcí) a vrátí $array[left]$.

Indukční krok: předpokládáme, že volání $f(array, left, (left + right)/2)$ po konečném počtu kroků vrátí minimum z množiny $L = \{array[left], \ldots, array[(left + right)/2]\}$, a že volání $f(array, (left + right)/2 + 1, right)$ po konečném počtu kroků vrátí minimum z množiny $R = \{array[(left+right)/2+1], \ldots, array[right]\}$; chceme dokázat, že volání $f(array, left, right)$ po konečném počtu kroků vrátí minimum z množiny $A = \{array[left], \ldots, array[right]\}$.

Konečnost počtu kroků je zjevná: kromě rekurzivních volání (která podle indukčního předpokladu skončí v konečném počtu kroků) se vykoná ještě první a poslední řádek. Oba neobshují žádné cykly ani volání funkcí, proto skončí v konečném počtu kroků. Ad parciální správnost: platí $A = L \cup R$, funkce f vrátí to menší z minim L a R, takže nutně musí vrátit minimum z A.

Indukcí dokažte následující výrok: pokud $left$ a $right$ jsou parametry funkce f a platí $left \leq right$ a pokud pole $array$ obsahuje hodnotu $elem$ alespoň na jednom z indexů mezi $left$ až $right$ včetně, pak volání $f(array, elem, left, right)$ v konečném počtu kroků vrátí hodnotu $true$, jinak $false$.

```
boolean f(int[] array, int elem, int left, int right) {
  if (left == right)
    return array[left] == elem;
  boolean l = f(array, elem, left, (left + right) / 2);
                          // celociselne deleni, zaokrouhluje dolu
  boolean r = f(array, elem, (left + right) / 2 + 1, right);
  return l || r;
}
```

Dlouhou vodorovnou čarou oddělte důkaz prvního a druhého indukčního kroku, u druhého indukčního kroku zřetelně napište, jak vypadá indukční předpoklad a co se z něj snažíte dokázat.

První krok: máme dokázat, že pro $left = right$ vrátí volání $f(array, elem, left, right)$ po konečném počtu kroků jestli $elem \in \{array[left]\}$. To zjevně platí: funkce

se vrátí hned na prvním řádku (na kterém nejsou žádné cykly ani volání dalších funkcí) a vrátí zda $elem = array[left]$.

Indukční krok: předpokládáme, že volání $f(array, left, (left + right)/2)$ po konečném počtu kroků vrátí zda $elem \in L$, kde $L = \{array[left], \ldots, array[(left + right)/2]\}$, a že volání $f(array, (left + right)/2 + 1, right)$ po konečném počtu kroků vrátí zda $elem \in R$, kde $R = \{array[(left+right)/2+1], \ldots, array[right]\}$; chceme dokázat, že volání $f(array, left, right)$ po konečném počtu kroků vrátí zda $elem \in A$, kde $A = \{array[left], \ldots, array[right]\}$.

Zjevně platí $A = L \cup R$, platí tedy $elem \in A \Leftrightarrow elem \in L \vee elem \in R$, což je přesně to, co funkce f vrací. Konečnost je taky zjevná: druhý a třetí řádek skončí v konečném čase podle indukčního předpokladu, první ani čtvrtý řádek neobsahují žádný cyklus ani volání funkce.

Napište, jakou hodnotu vrací průměrně funkce r pro daná n a k. Napište zřetelně postup vašeho odvození. O typu `double` uvažujte jako o typu s neomezenou přesností.

```
double r(int n, int k) {
  Random rnd = new Random();
  double[] b = new double[n];
  for (int i = 0; i < k; i++)
    b[rnd.nextInt(n)] = 1.0; //nahodne cislo 0..n-1
  double s = 0.0;
  for (int i = 0; i < n; i++)
    s += b[i];
  return s;
}
```

Nadefinujme si vektor náhodných veličin (indikátorů) tak, že $B_i = 1$, pokud je po projití prvního for cyklu v $b[i]$ jednička, a $B_i = 0$ jinak. Pravděpodobnost, že v $b[i]$ je nula (tzn. že $B_i = 0$), je $Pr\{b[i] = 0\} = \left(\frac{n-1}{n}\right)^k$. Průměrná hodnota sumy všech prvků v poli je pak

$$E\left[\sum_{i=0}^{n-1} B_i\right] = \sum_{i=0}^{n-1} E[B_i] = \sum_{i=0}^{n-1}(0 \cdot Pr\{b[i] = 0\} + 1 \cdot Pr\{b[i] = 1\}) =$$

$$= n \cdot \left(1 - \left(\frac{n-1}{n}\right)^k\right)$$

Popište, co vytiskne funkce *cp*. Napište, kolik řádků vytiskne pro obecná n a k. Demonstrujte své závěry na příkladu volání $cp(4, 2)$.

```
void cp(int n, int k) {
  cp2(new int[n], k, 0);
}

void cp2(int[] n, int k, int i) {
  if (i == n.length) {
    for(int a: n)
```

```
      System.out.print(a);  //pokracuje na radce
    System.out.println();   //konec radky
    return;
  }
  if(i < n.length-k) {
    n[i] = 0;
    cp2(n, k, i + 1);
  }
  if(k > 0) {
    n[i] = 1;
    cp2(n, k-1, i + 1);
  }
}
```

Funkce vytiskne všechny $n$ciferné řetězce složené z k jedniček a $n - k$ nul. Těch je stejně jako $k$prvkových podmnožin $n$prvkové množiny, tedy $\binom{n}{k}$.

Napište tvrzení, které bude platit pro návratovou hodnotu metody f, pokud tato metoda na vstupu obdrží celočíselné pole délky n, kde n je celé číslo větší než nula.

```
int f(int[] array) {
  int i = 0;
  for (int j = 1; j < array.length; j++)
    if (array[i] > array[j])
      i = j;
  return i;
}
```

Funkce f vrátí index prvního minima v poli $array$.

Funkce f vrátí i takové, že $\forall j \in \{1, \ldots, array.length - 1\} : array[i] < array[j] \vee (array[i] = array[j] \wedge i \leq j)$.

Napište tvrzení, které bude platit pro návratovou hodnotu metody f, pokud tato metoda na vstupu obdrží celočíselné pole délky n, kde n je celé číslo větší než nula.

```
int f(int[] array) {
  int i = 0;
  for (int j = 1; j < array.length; j++)
    if (array[i] < array[j])
      i = j;
  return i;
}
```

Funkce f vrátí index prvního maxima v poli $array$.

Funkce f vrátí i takové, že $\forall j \in \{1, \ldots, array.length - 1\} : array[i] > array[j] \vee (array[i] = array[j] \wedge i \leq j)$.

Čtvercová matice M o rozměru $k \times k$ obsahuje kladné a záporné prvky. Prvek na pozici (i, j) je kladný, pokud číslo $i + j$ je sudé, jinak je záporný. Algoritmus A

prochází po řádcích odshora dolů celou matici M. Pokud je prvek na pozici (i, j) kladný, provede s ním algoritmus A právě $k + i$ specifických operací. Pokud je prvek na pozici (i, j) záporný, provede s ním algoritmus A pouze k specifických operací. Indexy řádků i sloupců začínají od 0.

Napište odvození vztahu, který pro dané sudé k určí, kolik specifických operací algoritmus A provede v matici M o rozměru $k \times k$.

Matice počtu prováděných operací vypadá takto:

$$\begin{pmatrix} k+0 & k & \dots & k+0 & k \\ k & k+1 & \dots & k & k+1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ k+(k-2) & k & \dots & k+(k-2) & k \\ k & k+(k-1) & \dots & k & k+(k-1) \end{pmatrix}$$

Celkový počet operací na sudých sloupcích je

$$\frac{k}{2} \cdot k + \sum_{a=0}^{k/2-1} k + 2a = \frac{k^2}{2} + \frac{[k + (k + k - 2)] \cdot \frac{k}{2}}{2} = \frac{2k^2}{4} + \frac{3k^2 - 2k}{4} = \frac{5k^2 - 2k}{4},$$

celkový počet operací na lichých sloupcích je

$$\frac{k}{2} \cdot k + \sum_{a=0}^{k/2-1} k + 2a + 1 = \frac{k^2}{2} + \frac{[k + 1 + (k + k - 1)] \cdot \frac{k}{2}}{2} = \frac{2k^2}{4} + \frac{3k^2}{4} = \frac{5k^2}{4}.$$

Sudých i lichých řádků je $\frac{k}{2}$, celkový počet operací na všech sloupcích je proto

$$\frac{k}{2}\left(\frac{5k^2 - 2k}{4} + \frac{5k^2}{4}\right) = \frac{5}{4}k^3 - \frac{1}{4}k^2.$$

Alternativně, sčítáno po řádcích:

$$\sum_{i=0}^{k-1}\left(\frac{k}{2}k + \frac{k}{2}(k + i)\right) = \frac{k}{2}\sum_{i=0}^{k-1}(2k + i) = k^3 + \frac{k}{2}\sum_{i=0}^{k-1} i =$$
$$= k^3 + \frac{k}{2} \cdot \frac{[0 + (k - 1)]k}{2} = \frac{4k^3}{4} + \frac{k^3 - k^2}{4} = \frac{5}{4}k^3 - \frac{1}{4}k^2.$$

Když je procedura P zavolána s kladným celočíselným parametrem n, nejprve vytvoří $(n - 1)$ nových diskových souborů a nakonec zavolá sama sebe s parametrem $(n - 1)$. Pokud je P zavolána se záporným parametrem nebo s nulou, neprovede nic a skončí. Víme, že velikost každého nového diskového souboru je 200 B. Procedura P byla zavolána s neznámou hodnotou parametru n a po jejím skončení na disku ubylo 65 000 Bytů volného místa.

Určete hodnotu neznámého parametru n, napište, jak jste odvozovali příslušné vztahy.

Podle zadání ubyde po volání $P(n)$ $\sum_{k=1}^{n}(k-1) \cdot 200$ B volného místa. Tedy $65000 = \sum_{k=1}^{n}(k-1) \cdot 200 = \frac{(0+n-1)\cdot n}{2} \cdot 200 = (n^2 - n) \cdot 100$, $n^2 - n - 650 = 0$ a $n = 26$.

Funkce `f` zpracovává jednosměrný spojový seznam definovaný datovými typy `LinkedList` a `Node`. Popište jednou větou, co vrací funkce `f`, a implementujte ji znovu tak, aby pracovala co nejefektivněji.

```
class Node {
  int value; //Hodnota ulozena v prvku spojoveho seznamu
  Node next; //Odkaz na nasledujici prvek v seznamu
}

class LinkedList {
  Node head; //Odkaz na prvni prvek spojoveho seznamu
}

int f(LinkedList l) {
  Node mark = l.head;
  int t = 0;
  while (mark != null) {
    Node x = f1(l.head, mark);
    t = t + x.value;
    mark = x.next;
  }
  return t;
}

Node f1(Node n, Node mark) {
  if (n == mark)
    return n;
  else
    return f1(n.next, mark);
}
```

Funkce vrací součet prvků v zadaném spojovém seznamu.

```
int f(LinkedList l) {
    int r = 0;
    for (Node t = l.head; t != null; t = t.next) r += t.value;
    return r;
}
```

Funkce `f` zpracovává obousměrný spojový seznam definovaný datovými typy `DoubleLinkedList` a `Node`. Popište jednou větou, co vrací funkce `f`, a implementujte ji znovu tak, aby pracovala co nejefektivněji.

```
class Node {
  int value; //Hodnota ulozena v prvku spojoveho seznamu
  Node next; //Odkaz na nasledujici prvek v seznamu
  Node prev; //Odkaz na predchazejici prvek v seznamu
}

class DoubleLinkedList {
```

```
  Node head; //Odkaz na prvni prvek spojoveho seznamu
  Node tail; //Odkaz na posledni prvek spojoveho seznamu
}

int f(DoubleLinkedList l) {
  Node n = l.head;
  Node mark = l.tail;
  int t = 1;
  while (mark != null) {
    Node x = f1(l.head, mark);
    t = t * x.value;
    mark = mark.prev;
  }
  return t;
}

Node f1(Node n, Node mark) {
  if (n == mark)
    return n;
  else
    return f1(n.next, mark);
}
```

Funkce vrací součin prvků v zadaném spojovém seznamu.

```
int f(LinkedList l) {
    int r = 1;
    for (Node t = l.head; t != null; t = t.next) r *= t.value;
    return r;
}
```