Přímočaré řešení

Pro každou hodnotu parametru k spočteme zvlášť cenu celého procesu a pak vybereme takovou hodnotu k, pro níž je cena nejmenší.
Činnost algoritmu reprezentujeme stromem rekurze. Bez dobré představy stromu rekurze je obtížné úlohu srozumitelně řešit. Každý vnitřní uzel stromu odpovídá jednomu dělení určitého úseku na menší úseky a následnému sestavení řešení z výsledků nad těmito menšími úseky. Každý list stromu představuje (triviální) činnost algoritmu nad úsekem délky 1, za níž se nic neplatí.

Celková cena stromu je pak dána rekurentním vztahem
cena stromu = cena kořene + součet cen jednotlivých podstromů kořene
(protože každý podstrom představuje vlastně kompletní činnost algoritmu nad úsekem pole odpovídajícím kořeni tohoto podstromu) .
Kdybychom měli zaručeno, že délka aktuálního úseku je *vždy* beze zbytku dělitelná na k stejně velkých částí (to jest, kdyby délka celého pole byla mocninou k), pak bychom mohli napsat funkci, která počítá cenu stromu např. takto:

```
long cost(long k, long M) {
  if ((M == 0) || (M == 1))
    // problems of 0 or 1 size have no cost, also recursion ends here
    return 0;

  // divide the problem into k subproblems
  long M1 = M/k;            // size of the subproblems
  return (a*M*M+b*M+c)      // cost of division and later results merging
         + k*cost(k, M1);   // total cost of all subproblems
}
```

Protože ale délky úseků nebudou většinou takto výhodné, budeme muset rozdělit úsek délky M na k_1 úseků délky M_1 a k_2 úseků délky M_1+1. Tím zachováme pravidlo, že délky každých dvou kratších úseků pří dělení jednoho úseku délky M se liší nejvýše o 1. Bude tedy platit

$k = k_1 + k_2$,

$M = k_1 * M_1 + k_2 * (M_1+1)$.

Pro dané M a k jsou hodnoty k_1 , k_2, M_1 určeny jednoznačně (rozmyslete!):

$M_1 = \lfloor M/k \rfloor$,

$k_1 = k*(M_1+1) - M$,

$k_2 = k - k_1$.

Předchozí funkci nyní snadno rozšíříme pro uvedený případ kratších úseků různé délky:

```
long cost(long k, long M) {
  if ((M == 0) || (M == 1))
    // problems of 0 or 1 size have no cost, also recursion ends here
    return 0;

  long M1 = M/k;            // size of the subproblems
  if (k*M1 == M)
    // equal size of all subproblems
    return (a*M*M+b*M+c) + k*cost(k, M1);

  // different sizes (max by 1) of particular subproblems:
  long k1 = k*(M1+1)-M;  // number of bigger subproblems of size M1+1
  long k2 = k-k1;        // number of smaller subproblems of size M1
                         // obviously: k1+k2 = k
  return (a*M*M+b*M+c) + k1*cost(k, M1) + k2*cost(k, M1+1);
}
```

Když nyní funkci `cost` předáme jako parametr délku celého pole N, vrátí tato funkce celkovou cenu práce algoritmu A nad polem délky N při pevném parametru větvení *k*. Zbývá jen postupně volat tuto funkci s různými hodnotami parametru *k* a vybrat tu nejvýhodnější.
Otázka celkového počtu vnitřních uzlů stromu je velmi jednoduchá. Stačí opět spočítat celkovou cenu stromu pro optimální hodnotu *k* s tím, že každý vnitřní uzel bude mít cenu právě 1. Cena stromu pak bude rovna součtu cen vnitřních uzlů a v tomto případě zároveň i jejich počtu. Nastavíme tedy parametry ceny
$a = 0$, $b = 0$, $c = 1$ a zavoláme naposled funkci `cost`. Přiložený program v Javě dokumentuje uvedené úvahy.

```
package alg;

import java.io.*;
import java.util.StringTokenizer;

public class A02pub {

static long a, b, c;    // cost coefficients

static long cost(long k, long M) {
  if ((M == 0) || (M == 1))
    // problems of 0 or 1 size have no cost, also recursion ends here
    return 0;

  long M1 = M/k;         // size of the subproblems
  if (k*M1 == M)
    // equal size of all subproblems
    return (a*M*M+b*M+c) + k*cost(k, M1);

  // different sizes (max by 1) of particular subproblems
  long k1 = k*(M1+1)-M;  // number of bigger subproblems of size M1+1
  long k2 = k-k1;        // number of smaller subproblems of size M1
                         // obviously: k1+k2 = k
  return (a*M*M+b*M+c) + k1*cost(k, M1) + k2*cost(k, M1+1);
}

public static void main(String[] args) throws IOException {
  // read the values
  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
  StringTokenizer st = new StringTokenizer(br.readLine());
  long N = Long.valueOf(st.nextToken());
  a = Long.valueOf(st.nextToken());
  b = Long.valueOf(st.nextToken());
  c = Long.valueOf(st.nextToken());
  int k1 = Integer.valueOf(st.nextToken());  // not to confuse with k1 and k2
  int k2 = Integer.valueOf(st.nextToken());  // in the cost function :-)

  // search for minimum cost
  long cost;
  long mincost = Long.MAX_VALUE;
  int kmin=0;
  for(int k = k1; k <= k2; k++) {
    cost = cost(k,N);
    if (cost < mincost) {
      mincost = cost;
      kmin = k;
    }
  }

  // results
  a = 0; b = 0; c = 1;     // reset cost params to find no of internal nodes
  System.out.printf("%d %d %d\n", kmin, mincost, cost(kmin, N));
}

}
```