Solution

We proceed by scanning through the classrooms, we can compute a list of gaps: blocks of consecutive empty classrooms (the zeros). Let $l_1, l_2, \ldots, l_k$ be the lengths of these gaps. For example, consider the following input:

10001001001000

Then here, the gaps are: 3,2,2,3. (It would be useful to have some sort of function that outputs the maximum and minimum of this list)

There are a couple of cases. Usually, when a new student comes you want to place it in the **middle** of gaps. Say we want to put it in the first gap:

1**000**1001001000⇒1**0x0**1001001000

However if the gap is towards the **beginning or end**, you'd have to adjust it accordingly:

10010010010**00**⇒10001001001**00x**

There is an additional case where sometimes you may want to put **both** students in the same gap, you would need to put one of them ⅓ of the way through and another ⅔ of the way:

10001**00**1001000⇒10001**xx**1001000

Since it's very difficult to figure out which combination is the best, we simply try all of them and find the largest D possible.

C++ (Github for better readability)

```
#include <iostream>
#include <fstream>
using namespace std;

// Returns size of largest gap between two 1s and also the index where it
starts
int find_largest_interior_gap(string s, int &gap_start)
{
  int biggest_gap = 0, current_start = -1, N = s.length();
  for (int i=0; i<N; i++)
    if (s[i] == '1') {
      if (current_start!=-1 && i-current_start > biggest_gap) {
      biggest_gap = i-current_start;
      gap_start = current_start;
      }
      current_start = i;
    }
  return biggest_gap;
}

// Returns size of smallest gap between two 1s
int find_smallest_interior_gap(string s)
{
  int smallest_gap = 1000000000, current_start = -1, N = s.length();
  for (int i=0; i<N; i++)
    if (s[i] == '1') {
      if (current_start!=-1 && i-current_start < smallest_gap) smallest_gap
= i-current_start;
      current_start = i;
    }
  return smallest_gap;
}

// Outputs the smallest gap after adding a student into the largest gap
int try_student_in_largest_gap(string s)
{
  int gap_start, largest_gap = find_largest_interior_gap(s, gap_start);
  if (largest_gap >= 2) {
    s[gap_start + largest_gap / 2] = '1';
    return find_smallest_interior_gap(s);
  }
  return -1; // no gap!
}

int main(void)
{
  ifstream fin ("socdist1.in");
  int N;
  string s, temp_s;
```

```
  fin >> N >> s;
  ofstream fout ("socdist1.out");
  int answer = 0;

  // Possibility 1. put two students in largest interior gap
  int gap_start, largest_gap = find_largest_interior_gap(s, gap_start);
  if (largest_gap >= 3) {
    temp_s = s;
    temp_s[gap_start + largest_gap / 3] = '1';
    temp_s[gap_start + largest_gap * 2 / 3] = '1';
    answer = max(answer, find_smallest_interior_gap(temp_s));
  }

  // Possibility 2. students at both ends
  if (s[0] == '0' && s[N-1] == '0') {
    temp_s = s; temp_s[0] = temp_s[N-1] = '1';
    answer = max(answer, find_smallest_interior_gap(temp_s));
  }

  // Possibility 3. students at left + students in largest interior gap
  if (s[0] == '0') {
    temp_s = s; temp_s[0] = '1';
    answer = max(answer, try_student_in_largest_gap(temp_s));
  }

  // Possibility 4. Students at right + students in largest interior gap
  if (s[N-1] == '0') {
    temp_s = s; temp_s[N-1] = '1';
    answer = max(answer, try_student_in_largest_gap(temp_s));
  }

  // Possibility 5. Students at largest interior gap.  done twice.
  if (largest_gap >= 2) {
    temp_s = s; temp_s[gap_start + largest_gap / 2] = '1';
    answer = max(answer, try_student_in_largest_gap(temp_s));
  }

  fout << answer << "\n";
  return 0;
}
```

Java (Github for better readability)

```
import java.io.*;
import java.util.*;

class socdist1 {
    public static void main(String[] args) throws IOException{

        BufferedReader in = new BufferedReader(new
FileReader("socdist1.in"));
        PrintWriter out = new PrintWriter(new BufferedWriter(new
FileWriter("socdist1.out")));

        int N = Integer.parseInt( in.readLine() );
        String s = in.readLine();

        int currd = 100000000;
        int top1 = 1;
        int top2 = 1;
        int topAdd2 = 1;
        int gapstart = -1;
        for (int i = 0; i < N; i++) {
          boolean curr = s.charAt(i) == '1';
          if (curr) {
            if (gapstart == -1) {
              top1 = Math.max(top1, i);
              topAdd2 = Math.max(topAdd2, i/2);
              gapstart = i;
            }
            else {
              int j = (i-gapstart)/2;
              if (j >= top1) {
                top2 = top1;
                top1 = j;
              }
              else if (j > top2) {
                top2 = j;
              }
              topAdd2 = Math.max(topAdd2, (i-gapstart)/3);
              currd = Math.min(currd, i-gapstart);
              gapstart = i;
            }
          }
        }
        if (gapstart == -1) {
          topAdd2 = Math.max(topAdd2, N-1);
        }
        else {
          int j = N-gapstart-1;
          if (j >= top1) {
            top2 = top1;
```

```
            top1 = j;
          }
          else if (j > top2) {
            top2 = j;
          }
        }
        topAdd2 = Math.max(topAdd2, (N-gapstart-1)/2);
        out.println("" + Math.min(Math.max(Math.min(top1, top2), topAdd2),
currd));
        out.close();
    }
}
```

Python(Github for better readability)

```
f = open("socdist1.in", "r")

N = int(f.readline())
s = f.readline()

f.close()

#Returns size of largest gap between two 1s and also the index where it
starts
def find_largest_interior_gap(s):
      biggest_gap = 0
      current_start = -1
      gap_start = 0
      for i in range(N):
      if s[i] == "1":
            current_start = i
      else:
            if i-current_start > biggest_gap and current_start!=-1:
                  biggest_gap = i-current_start
                  gap_start = current_start
      return gap_start, biggest_gap + 1

#Returns size of smallest gap between two 1s
def find_smallest_interior_gap(s):
      s = s.strip("0")
      s = s[1:-1]
      s = s.split("1")
      smallest_gap = len(min(s)) + 1
      return smallest_gap
```

```
#Outputs the smallest gap after adding a student into the largest gap
def try_student_in_largest_gap(s):
      gap_start, largest_gap = find_largest_interior_gap(s)
      if largest_gap >= 2:
      s = s[:gap_start + largest_gap//2] + "1" + s[gap_start +
largest_gap//2 + 1:]
      return find_smallest_interior_gap(s)
      return -1 #no gap!

answer = 0

#Possibility 1. put two students in largest interior gap
gap_start, largest_gap = find_largest_interior_gap(s)
if largest_gap >= 3:
      temp_s = s
      temp_s = temp_s[:gap_start + largest_gap // 3] + "1" +
temp_s[gap_start + largest_gap // 3 + 1:]
      temp_s = temp_s[:gap_start + largest_gap * 2 // 3] + "1" +
temp_s[gap_start + largest_gap * 2 // 3 + 1:]
      answer = max(answer, find_smallest_interior_gap(temp_s))


#Possibility 2. students at both ends
if s[0] == "0" and s[-1] == "0":
      temp_s = s
      temp_s = "1" + temp_s[1:-1] + "1"
      answer = max(answer, find_smallest_interior_gap(temp_s))


#Possibility 3. students at left + students in largest interior gap
if s[0] == "0":
      temp_s = s
      temp_s = "1" + temp_s[1:]
      answer = max(answer, try_student_in_largest_gap(temp_s))


#Possibility 4. Students at right + students in largest interior gap
if s[-1] == "0":
      temp_s = s
      temp_s = temp_s[:-1] + "1"
      answer = max(answer, try_student_in_largest_gap(temp_s))


#Possibility 5. Students at largest interior gap.  done twice.
if largest_gap >= 2:
      temp_s = s
      temp_s = temp_s[:gap_start + largest_gap // 2] + "1" +
temp_s[gap_start + largest_gap // 2 + 1:]
      answer = max(answer, try_student_in_largest_gap(temp_s))
```

```
f = open("socdist1.out", "w")

f.write(str(answer))

f.close()
```