Question 4

```
#include <vector>
#include <iostream>
#include <cstdio>
using namespace std;

class Disjoint {
  public:
    Disjoint(int nelements);
    int Union(int s1, int s2);
    int Find(int element);
    void Print();
  protected:
    vector <int> links;
    vector <int> ranks;
};
```

I am running an instance of a **Disjoint** data structure, which is to the right. I do a bunch of operations on it, and when I call **Print()**, it prints the following:

```
Elts:   0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
Links: -1  0 -1  2  0  2  2  2  0  5  2 -1 -1 17 17 11 17  2 17 17
Ranks:  2  1  3  1  1  2  1  1  1  1  1  2  1  1  1  1  1  2  1  1
```

Answer the following questions:

- **A**: How many disjoint sets did I start with when I first started running the program?
- **B**: How many disjoint sets are there now?
- **C**: How many times has **Union()** been called?
- **D**: Union-by-size is most definitely not the implementation here. Why?
- **E**: What are the sizes of each set?
- **F**: Suppose I want to prove that the implementation uses path compression. I can do that by calling either **Union()** or **Find()**, and then **Print()**. Tell me which call it is (**Union()** or **Find()**), what the parameter(s) should be, and how the **Print()** statement will prove whether path compression is being implemented. I want *specifics* here, not vague junk like "call **Union()** on two sets and the **ranks** fields will tell you whether path compression is being used." That answer is a zero.

Question 5

- **A:** If a node is at index *i* in a heap, at what index is the node's parent?
- **B:** If a node is at index *i* in a heap, at what indices are the node's children?
- **C:** Suppose a heap is in the vector {8, 27, 66, 29, 58, 93, 71, 36, 78, 75}. What is the vector after you call **Push(9)**?
- **D:** Suppose a heap is in the vector {8, 27, 66, 29, 58, 93, 71, 36, 78, 75}. What is the vector after you call **Pop()**?

Question 6

Write a program that reads words on standard input, and prints them sorted by the average of their ASCII character values (in ascending order). Print one word per line. If two strings have the same averages, then print the one that you read in first before the other one. You may not make any assumptions on the number or words or the maximum word size.

Examples:

```
UNIX> echo bbb abb aaa baa | a.out
aaa
baa
abb
bbb
UNIX> echo xxxx xx wy ww | a.out
ww
xxxx
xx
wy
UNIX> echo aaaa bb aa bbbb a b aaaaa| a.out
aaaa
aa
a
aaaaa
bb
bbbb
b
UNIX>
```