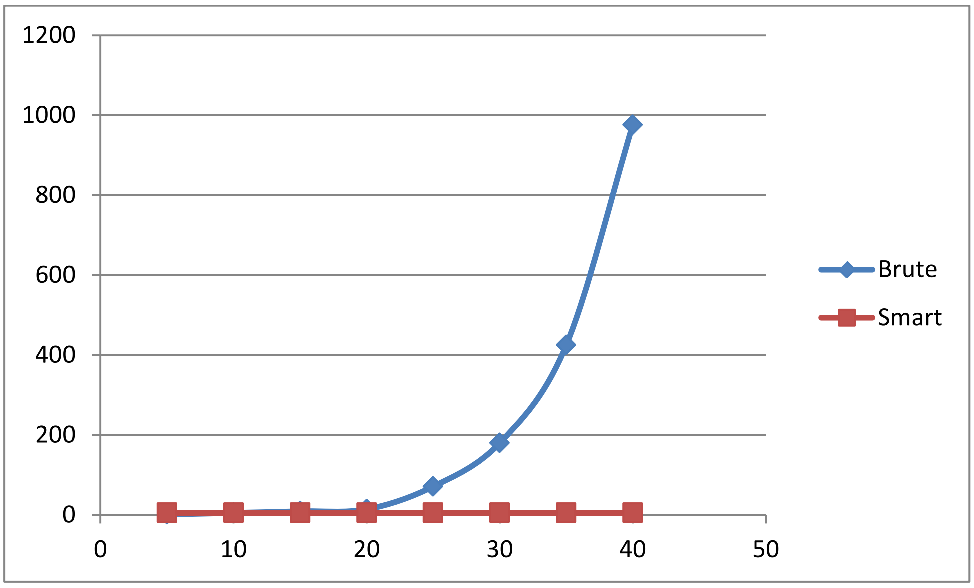
Timothée Guérin
260447866

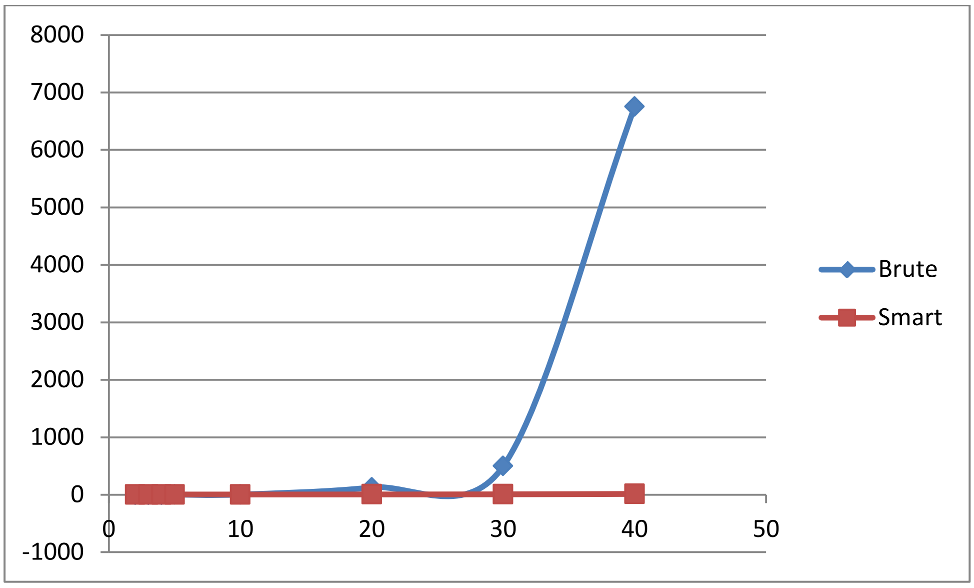
Question 1:

Question 1.c

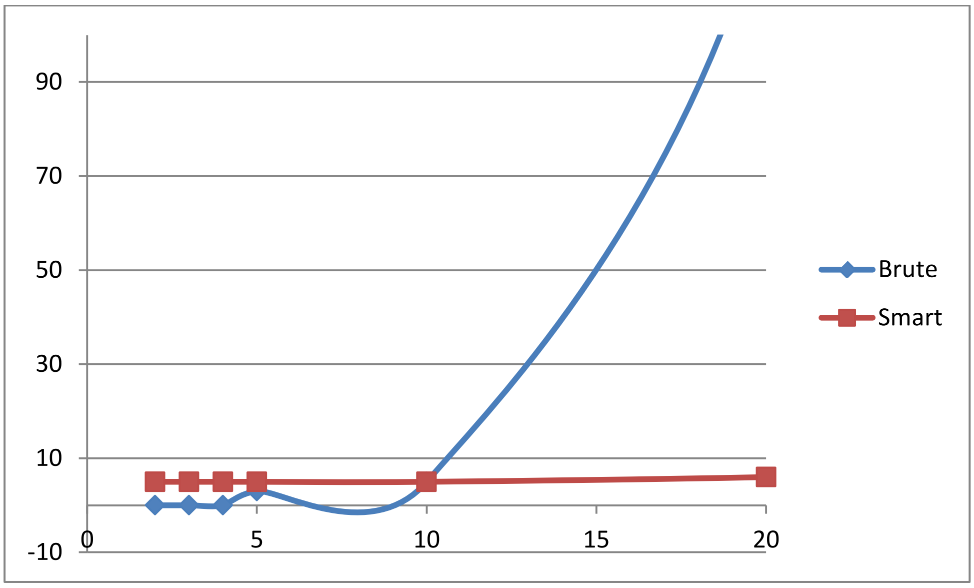
Changing the number of variable from 5 to 40 with 20 disjunctions:



Changing the number of disjunctions from 2 to 40 with 20 variables:



Zooming on the [2, 20] interval:



As we can see with this as we increase the number of variable or the number of disjunctions the brute force algorithm take exponentially more time. The smart algorithm is however always quite similar if we change the number of variable and increase slightly with the number of disjunction.

But we can see that with the graph three with a small number of variable the brute force algorithm is faster as there are not so many combination to generate(2^n)

Question 2:

Question 2.a

- **MakeQueue**:

```
makeQueue
{
    Queue A =  newQueue()
    Stack B = new Stack() // head of the queue
    Stack C = new Stack() // tail of the queue
}
```

- **EnQueue**:

```
enQueue( Queue A, object o)  // object o is the object we want to enqueue
{
    while(!B.isEmpty())
    {
        Object x=B.pop()
        C.push(x)
    }
    B.push(o);
    While(!C.isEmpty())
    {
        Object x = C.pop()
        B.push(x)
    }
}
```

- **DeQueue**:

```
Dequeue(Queue A)
{
    If(B.isEmpty())
    { //error
        Exit(0)
    }
    Return B.pop()
}
```

Question 2.b

Notice that the Enqueue method is quite costly in terms of running time compare to Dequeue which is a little bit more efficient.
We can rewrite these methods to improve their time bound by using amortization.
Our makeQueue function is the same as above

Enqueue:

```
enQueue(Queue A, Object o)
{
    Push(B, o)
}
```

enQueue runs in O(1).

__Dequeue__:
To improve this method let us use the fact that we can run it in O(1) if one of the stacks is empty

```
deQueue(Queue A)
{
    If(C.isEmpty())
    {
        While(!B.isEmpty())
        {
            Object o = B.pop()
            C.push(o)
        }
        Return B.pop()
    }
    Else
    {
        Return C.pop()
    }
}
```

If C is empty then dequeue runs in O(1) otherwise it runs in O(n)
Hence if we want to pop n times we get $T(n) = 1 + 1 + 1 \ldots + n$ (there is n ones)
In conclusion we get amortized time $T(n) = (n + n)/n = 2n/n$ which is O(1).

Hence we have proved that each queue operation can be performed in O(1) amortized time.

Question 3:

Suppose you have an undirected graph $G = (V, E)$, where $n = |V|$, and n is even. Prove that for all $n \geq 2$, if every $v\ E\ V$ has degree(v) ≥ n/2 then G is necessarily connected.

__Proof (by contradiction):__

The degree of a vertex is the number of edges connected to it.
A graph is connected if there is a path between all pairs of node

Assume that there exists some Graph $G = (V, E)$ with $n = |V|$ vertices and every vertex has degree $n/2$ but G is not connected.
For any vertex v E G, there exists an edge from v to at least $n/2$ other vertices.
Hence, the connected component* C that contains v consists of at least $1 + n/2$ vertices.
Now, because we assumed that G is not connected, let us consider some other vertex r which is not in C. There are at most $n - \left(1 + \frac{n}{2}\right) = \frac{n}{2} - 1$ vertices that are not in C. Hence there is at least $\frac{n}{2}$ edges from the vertex r and there must be at least one edge from r to a vertex in C which implies that r is in C and hence we have a contradiction.

In conclusion G is necessarily connected.

Connected component:

A connected component of an undirected graph is a subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the supergraph.