Andrés Felipe Losada 201631453

Santiago Rangel 201632011

Tarea 1

1.a.

1.b.

Se puede hacer la prueba del DO para verificar el programa:

1. P vale antes:
2. P sirve:
3. P invariante:

Por lo que,

1. Terminación

Este ciclo termina porque cada vez que toma una acción i se acerca mas a n o, si se cumple el ciclo n = 0 y por ende n nunca es mayor a 1 y se finaliza la iteración.

2.

F7: (+k| 1≤k≤n: 2 -k\*(1+√5)k) : (

3.

a) Hay que demostrar los teoremas para probar que el do sirve:

* P vale antes
* P sirve
* P invariante
* Hay terminación

**P vale antes:**

**P sirve:**

R1:

**P Invariante:**

**3.1**

**3.2**

**3.3**

**3.4** f[p] ≤ f[q] , m:= q {P}  
**3.5** f[p] ≥ f[q], m:= p {P}

**3.6** f[p] ≥ f[p+1] ∧ f[p+1] ≥ f[q], m:= p {P}

**3.7** f[p] ≥ f[p+1] ∧ f[p+1] ≥ f[q], m:= p {P}

b) Hay terminación porque la cota baja con cada iteración, en las primeras iteraciones del do p incrementa y q decrece. En los ifs después del R1 no decrece más la cota pero se busca el m, que es la aproximación más cercana a h.

c). Si contamos las asginaciones como las operaciones básicas podemos descartar la parte del código después del do ya que el número de asignaciones ahí no depende de y su máximo será una constante. Ahora si analizamos el do podemos darnos cuenta de que a medida que la diferencia entre p y q (es decir, entre a y b) incrementa, también el numero de asignaciones. Esto quiere decir que la complejidad crece con n que es a-b. También podemos darnos cuenta de que entre mayor sea la diferencia entre p y q, en la siguiente iteración p incrementa en razón a esta diferencia divido por 3. Por lo tanto T = θ(n/3).

4.

* S(n+1) = 2\*S(n) + 2\*S(n-1) … + 2\*S(1) +
* S(n) = 2(n-1) … S(1) +
* S(n+1) = 2\*S(n) + -
* S(n+1) - S(n) = - = 2n + 1
* (E – 3) = 0
* A\*+ Bn + C = 0
* S(1) = 1 = A3 + B + C
* S(2) = 6 = A9 + B2 + C
* S(3) = 23 = A27 + B3 + C

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 1 | 1 | 1 |
| 9 | 2 | 1 | 6 |
| 27 | 3 | 1 | 23 |

* A = 1, B = -1, C = -1
* S(n) = - 1 - n