Solució de l'Examen Parcial EDA 11/11/2019

Proposta de solució al problema 1

(a) $\Theta(\sqrt{n}\log n)$

(b) Calculem:

$$\lim_{n\to\infty}\frac{\log(\log n^2)}{\log n}=\lim_{n\to\infty}\frac{\log(2\log n)}{\log n}=\lim_{n\to\infty}\frac{\log 2+\log(\log n)}{\log n}=$$

$$=\lim_{n\to\infty}\frac{\log 2}{\log n}+\lim_{n\to\infty}\frac{\log(\log n)}{\log n}=\lim_{n\to\infty}\frac{\log(\log n)}{\log n}$$

Fem el canvi de variable $n=2^m$ i obtenim que l'anterior és igual a

$$\lim_{m\to\infty}\frac{\log(\log 2^m)}{\log 2^m}=\lim_{m\to\infty}\frac{\log m}{m}=0$$

Per tant únicament és cert que $\log(n)\in\Omega(\log(\log(n^2)))$

Proposta de solució al problema 2

(a) Retorna $f\circ g$, la composició de f amb g. El cost de *misteri* és el de la funció auxiliar *misteri_aux*, que ve descrit per la recurrència $T(n)=T(n-1)+\Theta(1)$, que té com a solució asimptòtica $T(n)\in\Theta(n)$.

(b) Retorna f^k. És a dir, una funció tal que $f^k(x)=\underbrace{f(f(\ldots(f}_{k}(x))))$. En funció de k, el seu cost ve donat per la recurrència $T(k)=T(k-1)+\Theta(1)$, que té com a solució $\Theta(k)$.

(c)

```
vector<int> misteri_2_quick(const vector<int>& f, int k) {
  if (k == 0) {
    vector<int> r(f.size());
    for (uint i = 0; i < f.size(); ++i) r[i] = i;
    return r;
  }
  else if (k%2 == 0) {
    vector<int> aux = misteri_2_quick(f,k/2);
    return misteri(aux,aux);
  }
  else {
    vector<int> aux = misteri_2_quick(f,k/2);
    return misteri(f, misteri(aux,aux));
  }
}
```

La recurrència que descriu el cost en temps d'aquesta funció és $T(k)=T(k/2)+\Theta(1)$, que té com a solució asimptòtica $\Theta(\log k)$.

Proposta de solució al problema 3

(a) És fàcil veure que la funció *max_suma* essencialment implementa una ordenació per selecció, que sabem que té cost en cas pitjor de $\Theta(m^2)$. L'única diferència és la línia on actualitzem *suma*, que triga temps constant i només s'executa m vegades. Per tant el cost total és $\Theta(m^2) + \Theta(m) = \Theta(m^2)$.

(b) Si entenem el codi anterior ens podem adonar que ordena el vector de major a menor i agrupa els enters consecutivament de dos en dos seguint aquest ordre. Per millorar l'eficiència, només cal ordenar el vector amb un *merge sort*, de manera que el cost sigui $\Theta(m \log m)$, i agrupar els enters consecutivament de dos en dos. El cost asimptòtic en temps seria de $\Theta(m \log m)$.

(c) Assumim que x_0 i x_1 són els dos nombres més grans de S i considerem una expressió que conté els productes $x_0 * y$ i $x_1 * z$, per certs $y, z \in S$. El que farem és reemplaçar aquests dos productes per $x_0 * x_1$ i $y * z$. Observem ara el següent: $(x_0 * x_1 + y * z) - (x_0 * y + x_1 * z) = x_0(x_1 - y) + (y - x_1)z = x_0(x_1 - y) - (x_1 - y)z = (x_0 - z)(x_1 - y) > 0$. L'últim pas és degut a que $x_0 > z$ i $x_1 > y$ ja que x_0 i x_1 són els elements majors de S, i són tots diferents. Per tant l'expressió original no era màxima ja que l'expressió resultant és major.

Anem a demostrar el resultat per inducció sobre m:

- *Cas base* ($m = 0$). L'algorisme és correcte ja que retorna una expressió que suma zero i per tant és òptima.
- *Pas d'inducció.* Sigui $m > 0$ i assumim la hipòtesi d'inducció: l'expressió màxima per un conjunt de $< m$ elements es pot obtenir ordenant els elements de major a menor i agrupant-los de dos en dos consecutivament. Si ordenem els m elements $x_0 > x_1 > x_2 > x_3 > \cdots > x_{m-1}$, sabem gràcies al resultat anterior que l'expressió òptima conté el producte $x_0 * x_1$ seguit d'una expressió formada amb els nombres $\{x_2, x_3, \ldots, x_{m-1}\}$. Aquesta expressió serà òbviament la major que puguem formar amb $\{x_2, x_3, \ldots, x_{m-1}\}$ i aplicant la hipòtesi d'inducció sabem que tindrà la forma $x_2 * x_3 + \cdots + x_{m-2} * x_{m-1}$. Per tant, l'expressió òptima és $x_0 * x_1 + x_2 * x_3 + \cdots + x_{m-2} * x_{m-1}$, com volíem demostrar.

Proposta de solució al problema 4

(a)

```
int f(const vector<int>& p, int l, int r){
  if (l + 1 ≥ r) return (p[l] ≤ p[r] ? l : r);
  else {
    int m = (l+r)/2;
    if (p[m] > p[m+1]) return f(p,m+1,r);
    else if (p[m−1] < p[m]) return f(p,l,m−1);
    else return m;
  }
```

```
}

pair<int,int> max_guany (const vector<int>& p) {
  return {f(p,0,p.size()-1), p.size()-1};
}
```

El cost de *max_guany* coincidirà amb el cost de la funció f. Per analitzar aquesta última, cal fixar-se que el seu cost ve donat per la recurrència $T(n) = T(n/2) + \Theta(1)$, d'on s'obté el cost de $\Theta(\log n)$.

(b)

```
int max_guany (const vector<int>& p, int k) {
  int m = p[k];
  for (int i = k - 1; i >= 0; --i)
    m = min(m,p[i]);

  int M = p[k];
  for (int i = k + 1; i < p.size (); ++i)
    M = max(M,p[i]);

  return M - m;
}
```

(c) Podem utilitzar un algorisme de dividir i vèncer. Donat un vector p, el partim en dues meitats, separades pel punt mig m. Recursivament, calculem el màxim guany possible si comprem i venem a la part esquerra del vector, i a continuació, també recursivament, calculem el màxim guany possible si comprem i venem a la part dreta del vector. Finalment, utilitzant la funció de l'apartat anterior, calculem el màxim guany d'un període que inclou el punt mig m (és a dir, comprem a la part esquerra i venem a la part dreta). El resultat final és el màxim dels tres guanys calculats.

Hem desenvolupat un esquema de dividir i vèncer on fem dues crides recursives de mida la meitat, i a continuació fem un treball lineal per calcular el màxim guany que inclogui el punt m. Per tant, la recurrència que determina el cost de la funció és: $T(n) = 2T(n/2) + \Theta(n)$, que té com a solució asimptòtica $\Theta(n \log n)$.

Nota: hi ha solucions més eficients no basades en dividir i vèncer.