Solució de l'Examen Parcial EDA 21/04/2023

Proposta de solució al problema 1

(a) El cas pitjor es dona quan s'efectuen totes les iteracions al bucle més extern sense trobar una solució. Centrem-nos, doncs, en aquest cas.

Sabem que el bucle extern farà n voltes. Per cadascuna d'aquestes voltes, el bucle intern en farà n. Per tant, el cos de bucle més intern s'executarà exactament n^2 vegades.

Aquest cos executarà dues instruccions *swap*, una comparació entre enters i dues crides a *suma*. Si no fos per aquestes dues crides, el cos tindria cost $\Theta(1)$. Una crida a *suma* involucra passar un vector per referència, inicialitzar la variable s a zero, retornar un enter (tot plegat cost $\Theta(1)$), així com un bucle que itera n vegades acumulant el valor dels elements de v a la variable s. Així doncs, el cost d'una crida a *suma* és $\Theta(n)$. Resumint, el cost del cos del bucle més intern és $\Theta(1) + 2\Theta(n) = \Theta(n)$.

Per tant, com que aquest s'executa n^2 vegades, tenim un cost total de $n^2 \cdot \Theta(n) = \Theta(n^3)$.

(b) La part del codi a completar és:

```
int x = v1[i] + dif/2;
```

Passem a analitzar el cost en cas pitjor d'una crida a *sol*. Com abans, el cas pitjor tindrà lloc quan executem totes les iteracions del bucle més extern.

La primera línia de *sol* efectua dues crides a *suma* i una resta, pel que té cost $\Theta(n)$. La segona línia té cost $\Theta(1)$, doncs només es fan operacions aritmètiques bàsiques i una comparació amb zero. En el cas pitjor, el bucle farà n voltes, i en cadascuna d'elles s'efectuarà una crida a *cerca* i altres operacions $\Theta(1)$. Així doncs, ja podem concloure que el cost serà n vegades el cost d'una crida a *cerca*.

La funció *cerca* és una funció recursiva que, en cas pitjor, efectua tot d'operacions aritmètiques de cost $\Theta(1)$ i dues crides recursives. Com que m és el punt mig del vector, ens n'adonem que les crides recursives són de mida la meitat. Així doncs, la recurrència que descriu el cost d'aquesta funció és $C(n) = 2 \cdot C(n/2) + \Theta(1)$. Si apliquem el teorema mestre per a recurrències divisores del tipus $C(n) = a \cdot C(n/b) + \Theta(n^k)$, podem identificar que $a = b = 2$, $k = 0$ i, per tant $\alpha = \log_2 2 = 1$. Com que $\alpha > k$, tenim que la recurrència té solució $C(n) \in \Theta(n^\alpha) = \Theta(n)$.

Per tant, el cost total serà $n \cdot \Theta(n) = \Theta(n^2)$.

(c) Per tal de millorar el cost, farem que la crida a *cerca* sigui més eficient. En concret, a la funció *sol*, just abans de l'inici del bucle, ordenarem el vector v_2 amb un algorisme d'ordenació que ens garanteixi cas pitjor $n \log n$, com pot ser el *mergesort*. Una vegada ordenat, enlloc de la funció *cerca* podem utilitzar una cerca dicotòmica, que tindrà cost $\Theta(\log n)$.

Si ho comparem amb l'anàlisi de cost anterior, veiem que haurem d'afegir el cost de l'ordenació, i reemplaçar el cost de *cerca* pel cost de la cerca dicotòmica. Per tant, el cost total serà $\Theta(n \log n) + n \cdot \Theta(\log n) = \Theta(n \log n)$.

Proposta de solució al problema 2

(a) $n = 10, k = 2 \Rightarrow 10 = 2^3 + 2^1$

$n = 10, k = 3 \Rightarrow 10 = 2^2 + 2^2 + 2^1$

$n = 10, k = 4 \Rightarrow 10 = 2^2 + 2^1 + 2^1 + 2^1$

$n = 10, k = 5 \Rightarrow 10 = 2^1 + 2^1 + 2^1 + 2^1 + 2^1$

(b) Com que sabem que la representació en binari d'n és la representació com a potències de 2 amb el menor nombre de sumands, si k és menor que el nombre d'uns de la representació en binari no hi ha solució possible.

D'altra banda, és fàcil adonar-se que la representació d'n en potències de dos amb el major nombre de sumands és la suma d'n sumands 2^0. Així doncs, si $k > n$ tampoc hi haurà solució.

(c) Una possible solució és:

```
vector<int> pos_uns (int n) {
  vector<int> v;
  int pos = 0;
  while (n ≠ 0) {
    if (n%2 == 1) v.push_back(pos);
    ++pos;
    n = n/2;
  }
  return v;
}
```

Veiem que totes les operacions que fa la funció tenen cost $\Theta(1)$ (assumint que el *push_back* té cost $\Theta(1)$). Així doncs, el cost vindrà donat pel nombre de voltes que faci el bucle. Si n_0 és el valor inicial de la variable n, en acabar la primera iteració, n val com a molt $n_0/2$ (recordem que la divisió entera en C++ trunca cap a baix). En acabar la segona, val com a molt $n_0/2^2$, en acabar la tercera com a molt $n_0/2^3$ i, en general després de la iteració k-èssima valdrà com a molt $n_0/2^k$. Per tant, podem garantir que quan $n_0 < 2^k$ el bucle s'aturarà perquè n valdrà zero. Això passa quan $\log n_0 < k$, i per tant, podem garantir que el bucle donarà com a molt $\Theta(\log n_0)$ voltes. Com que n_0 era el valor inicial d'n, podem concloure que el cost és $\Theta(\log n)$.

(d) Una possible solució és:

```
void escriu_suma_potencies (int n, int k) {
  vector<int> uns = pos_uns(n);
  if (uns.size() > k or k > n)
    cout << "No hi ha solucio" << endl;
  else {
    priority_queue<int> Q;
    for (auto x : uns) Q.push(x);
    while (Q.size() < k) {
      int m = Q.top();
      Q.pop();
      Q.push(m−1);
      Q.push(m−1);
    }
    bool primer = true;
    while (not Q.empty()){
      if (not primer) cout << " + ";
      else primer = false;
      cout << "2^" << Q.top();
      Q.pop();
    }
    cout << endl;
  }
}
```