1:

```
def dcsqrt(lo, hi, target):
    mid = (lo + hi)/2
    if (round(mid * mid, 50) == target):
        return(round(mid, 5))
    if (mid * mid > target):
        return(dcsqrt(lo, mid-1, target))
    return(dcsqrt(mid+1, hi, target))
```

$c$ (bloco das linhas mid, if, return, if)

$T(\frac{n}{2})$ — return(dcsqrt(lo, mid-1, target))

$T(\frac{n}{2})$ — return(dcsqrt(mid+1, hi, target))

$\Downarrow$

$$T(n) = T\left(\frac{n}{2}\right) + C$$

Pelo Teorema Mestre: $n^{\log_2 1} = n^0 = 1$

$f(n) = C$ e $n^{\log_2 0} = 1$ (difere por K constante)

$\Downarrow$

$$f(n) = \theta(n^{\log_2 0}) = \theta(1)$$

$\Downarrow$

$$\boxed{T(n) = \theta(n^{\log_b a} \lg(n)) = \theta(\lg(n))}$$ $\square$

2: A) $1+2+3+\dots+n \overset{?}{=} \frac{n.(n+1)}{2}$

P(1): $1 = \frac{1.(1+1)}{2} = 1$ ✓

P(K): verdade $\Rightarrow \sum_{i=1}^{K} = \frac{K.(K+1)}{2}$

P(K+1): $\sum_{i=1}^{K} + (K+1) \overset{?}{=} \frac{(K+1).((K+1)+1)}{2}$

$\frac{K.(K+1)}{2} + (K+1) = \frac{K(K+1)+2K+2}{2} =$

$\frac{K^2+3K+2}{2}$ $\quad \Delta = 9-4.1.2 = 1$ $\quad K = \frac{-3\pm 1}{2} = -2$ ou $-1$

$\boxed{\frac{(K+1).(K+2)}{2} \overset{\checkmark}{=} \frac{(K+1)((K+1)+1)}{2}}$ □

2: B) $1^2+2^2+3^2+\dots+n^2 = \frac{n.(n+1).(2n+1)}{6}$

P(1): $1 = \frac{1.(1+1).(2.1+1)}{6} = \frac{6}{6} = 1$

P(K): True $\Rightarrow \sum_{i=1}^{K} i^2 = \frac{K.(K+1).(2K+1)}{6}$

P(K+1): $\sum_{i=1}^{K} i^2 + (K+1)^2 \overset{?}{=} \frac{(K+1)(K+2)(2K+3)}{6}$

$\frac{K.(K+1).(2K+1)}{6} + (K^2+2K+1) = \frac{2K^3+3K^2+K}{6} +$

$(K^2+2K+1) = \frac{2K^3+9K^2+13K+6}{6}$ *

$\frac{(K+1).(K+2).(2K+3)}{6} = \frac{(K^2+3K+2).(2K+3)}{6} =$

$\boxed{\frac{2K^3+9K^2+13K+6}{6} = *}$ □

2: C) $1^3+2^3+3^3+\dots+n^3 = (1+2+3+\dots+n)^2$

P(1): $1^3 = (1)^2$ ✓

P(K): $\sum_{i=1}^{K} i^3 = \left(\sum_{i=1}^{K} i\right)^2$ true

P(K+1): $\overset{\star}{\sum_{i=1}^{K} i^3 + (K+1)^3} \overset{?}{=} \overset{\star\star}{\left(\sum_{i=1}^{K+1} i\right)^2}$

★ $\left(\sum_{i=1}^{K} i\right)^2 + K^3+3K^2+3K+1$

★★ $\left(\overset{=y}{\sum_{i=1}^{K} i} + K+1\right)^2$

★ $\boxed{y^2+K^3+3K^2+3K+1}$

★★ $(y+K+1)^2$

★★ $y^2+yK+y+Ky+K^2+K+$
$y+K+1$

★★ $\boxed{y^2+K^2+2yK+2y+2K+1}$

$\Downarrow$

★ ≠ ★★ ⇒ não funciona

∀ caso.

2: D) $1+3+5+\dots+(2n-1) = n^2$

P(1): $1 = 1^2 = 1$ ✓

P(K): $1+3+\dots+(2K-1) = K^2$ true

P(K+1): $\underbrace{1+3+\dots+(2K-1)}_{K^2} + (2K+1) \overset{?}{=} (K+1)^2$

$\boxed{K^2+(2K+1) = K^2+2K+1 = (K+1)^2}$ □

2: E) $1^3+2^3+\dots+n^3 = \frac{n^2(n+1)^2}{4}$

P(1): $1^3 = \frac{1^2.(1+1)^2}{4} = 1$ ✓

P(K): $\sum_{i=1}^{K} i^3 = \frac{K^2(K+1)^2}{4}$ true

P(K+1): $\overset{\star}{\sum_{i=1}^{K} i^3 + (K+1)^3} \overset{?}{=} \overset{\star\star}{\frac{(K+1)^2(K+2)^2}{4}}$

★ $\frac{K^2(K+1)^2}{4} + (K+1)^3 = \frac{K^4+2K^3+K^2}{4} +$

$K^3+3K^2+3K+1 = \boxed{\frac{K^4+6K^3+13K^2+12K+4}{4}}$

★★ $\frac{(K+1)^2.(K+2)^2}{4} = \frac{(K^2+2K+1).(K^2+4K+4)}{4}$

$= \boxed{\frac{K^4+6K^3+13K^2+12K+4}{4}}$

$\Downarrow$

$\boxed{★ = ★★}$ □

③: def dijkstra(graph, vstd, cost, start):

```
c {  cost[start] = 0
     pq = [ ]
     heappush(pq, [cost[start], start])
(V) → while (pq):
log(E) →    v = heappop(pq)[1]
c {         if (vstd[v]):
                continue
            vstd[v] = True
(E)/(V) →   for u in graph[v]:
c {             if (cost[v] + u[1] < cost[u[0]]):
                    cost[u[0]] = cost[v] + u[1]
log(E) →            heappush(pq, [cost[u[0]], u[0]])
```

③: Analisando, podemos ver que ele fará $|V| \lg(|E|)$ no começo e $\frac{|E|}{|V|} \lg(|E|)$ no final. Note que ele só entra no último for $|V|$ vezes, daí, teremos:

$$|V| \lg(|E|) + \cancel{|V|} \cdot \frac{|E|}{\cancel{|V|}} \lg(|E|)$$

$$\boxed{O((|V| + |E|) \lg(|E|))}_{\square}$$

Corretude: Logo no começo, o menor caminho para o nó mais próximo é justamente o caminho do início para o nó mais próximo. Similarmente, no final, o menor caminho para o último nó é o justamente o menor dentre os que existem, em ambos os casos, Dijkstra funciona. Agora note que o sub-caminho do menor caminho também é o menor caminho, daí o crescimento do algoritmo mantém-se correto.

(4:) BFS:

A → while (queue.notEmpty()):

c { v = queue.pop()
if (visited[v]):
continue
visited[v] = True

B → for u in graph[v]
c { queue.push(u)

(4:) DFS: c { if (visited[v]):
return
visited[v] = True

B → for u in graph[v]:
A → dfs(graph, visited, u)

numa matriz, o custo de "B" é |V|, já que ele obrigatoriamente tem que percorrer toda linha da matriz |V| x |V|, e "A" será executado |V| vezes, já que ele está empilhando/enfileirando em todo elemento da linha, daí:

$$\boxed{O(|V| \cdot |V|) = O(|V|^2)}$$

com lista de adjacência, o custo de "B" é reduzido para $\frac{|E|}{|V|}$ que é a quantidade de conexões de cada nó, e "A" continuará percorrendo todos os |V| vertices, daí:

$$O\left(|V| + |V| \cdot \frac{|E|}{|V|}\right) = \boxed{O(|V| + |E|)}$$

(5) Teorema Mestre: $T(n) = aT\left(\frac{n}{b}\right) + f(n)$, $a \geq 1$, $b > 1$, $f(n) > 0$

$f(n)$ compara $n^{\log_b a}$

A) $T(n) = T(n/2) + 3 \Rightarrow a = 1, b = 2 \Rightarrow n^{\log_2 1} = 1$

$3 = \Theta(1) \Rightarrow \boxed{T(n) = \Theta(\lg(n))}$

B) $T(n) = 9T(n/3) + n \Rightarrow a = 9, b = 3, f(n) = n \Rightarrow n^{\log_3 9} = n^2$

$n = O(n^2) \Rightarrow \boxed{T(n) = \Theta(n^2)}$

C) $T(n) = 3T(n/2) + n^2 \Rightarrow a = 3, b = 2, f(n) = n^2 \Rightarrow n^{\log_2 3} = n^{1,5\ldots}$

$n^2 = \Omega(n^{\log_2 3})$, e $3 \cdot \left(\frac{n}{2}\right)^2 \leq c\, n^2$ onde $c < 1$

$\to \frac{3}{4} < 1$

$\Downarrow$

$\boxed{T(n) = \Theta(n^2)}$

D) $T(n) = 4T(n/2) + n^2 \Rightarrow a = 4, b = 2, f(n) = n^2 \Rightarrow n^{\log_2 4} = n^2$

$n^2 = \Theta(n^2) \Rightarrow \boxed{T(n) = \Theta(n^2 \lg(n))}$

E) Anulada!

(5) F) $T(n) = 16T(n/4) + n! \Rightarrow a = 16, b = 4, f(n) = n!$

$n^{\log_4 16} = n^2$, $\quad n! = \Omega(n^2)$ e $\frac{16n!}{24} \leq c\, n!$ onde $c < 1$

daí: $\boxed{T(n) = \Theta(n!)}$

G) $T(n) = 4T(n/2) + n/\lg(n) \Rightarrow a = 4, b = 2, f(n) = \frac{n}{\lg(n)}$

$n^{\log_2 4} = n^2$, $\quad \frac{n}{\lg(n)} = O(n^2) \Rightarrow \boxed{T(n) = \Theta(n^2)}$

(6:) def firstFor(n):

```
r { if(n<0):
        return
p → secondFor(n**2)
T(n-1) → firstFor(n-1)
```

---

```
def secondFor(K):
  r { if(K<0):
          return
  K → thirdFor(K)
T(K-1) → secondFor(K-1)
```

---

```
def thirdFor(i):
  { if (i < 0):
        return
    print("PAA é legal")
T(i-1) → thirdFor(i-1)
```

---

$\Downarrow$

ThirdFor é $O(i)$,

SecondFor é $T(K) = T(K-1) + K =$

$T(K-2) + K-1+K = T(K-2) + 2K-1 =$

$T(1) + K^2 - (K-1)$, daí: $O(K^2)$

FirstFor é $T(n) = T(n-1) + p =$

$T(1) + np - (p-1)$, daí: $O(np)$,

mas $p$ custa $O(K^2)$, e K é chamado com $n^2$, daí: $O(n \cdot n^4) = O(n^5)$. $\square$