# Algoritmos de Busca Intervalar Ortogonais

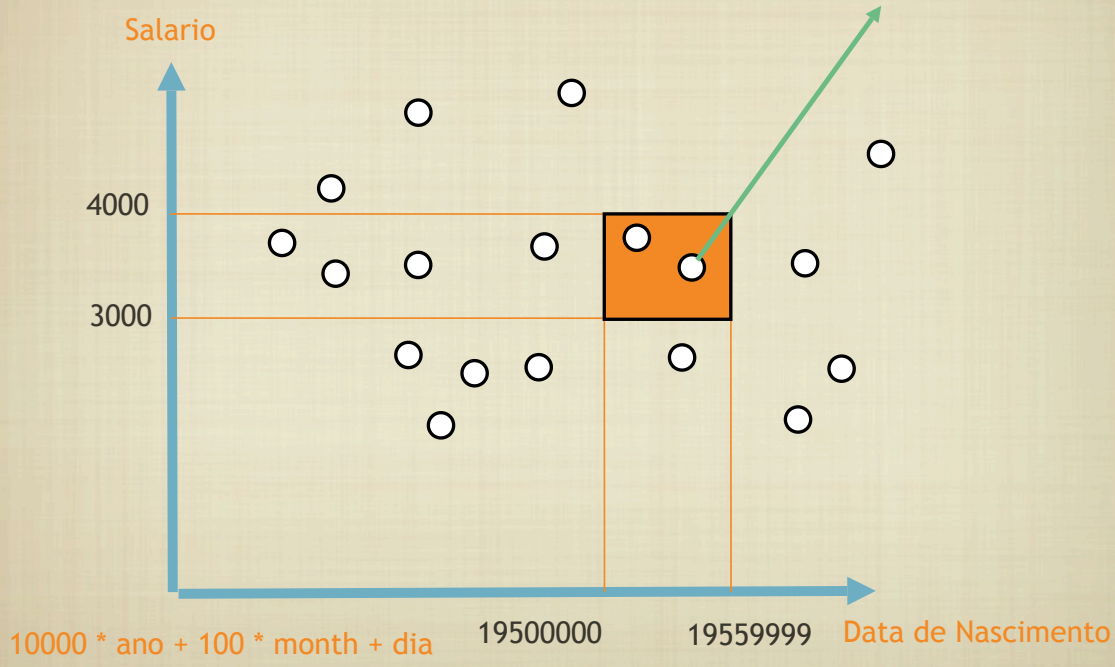
João Comba

## Problema

| Nome | Nascimento | Salario |
|---|---|---|
| Jose da Silva | 19/08/1954 | 3500 |
| Joao da Silva | 21/03/1936 | 7000 |
| Luiz da Silva | 22/10/1981 | 1000 |
| ... | ... | ... |

Q: Reportar todos funcionarios nascidos entre 1950 e 1955 que recebem entre 3000 e 4000 por mes

# INTERPRETACAO

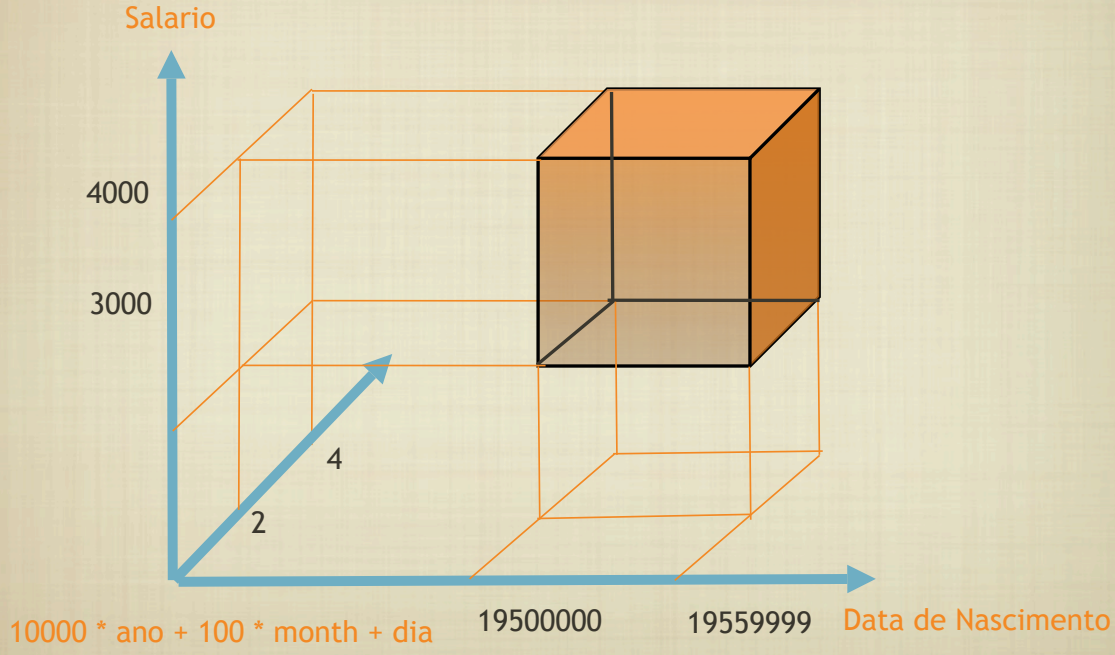



# PROBLEMA

| NOME | NASCIMENTO | SALARIO | FILHOS |
|---|---|---|---|
| JOSE DA SILVA | 19/08/1954 | 3500 | 3 |
| JOAO DA SILVA | 21/03/1936 | 7000 | 0 |
| LUIZ DA SILVA | 22/10/1981 | 1000 | 2 |
| ... | ... | ... | |

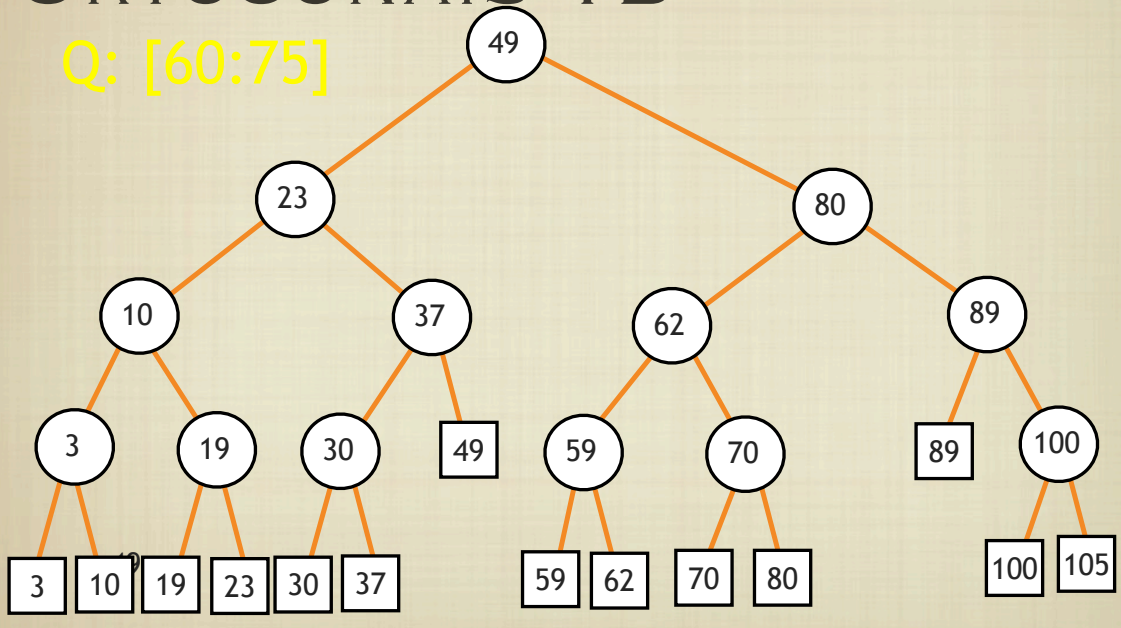
Q: Reportar todos funcionarios nascidos entre 1950 e 1955 que recebem entre 3000 e 4000 por mes e que tenham entra 2 e 4 filhos

# Interpretacao Geometrica

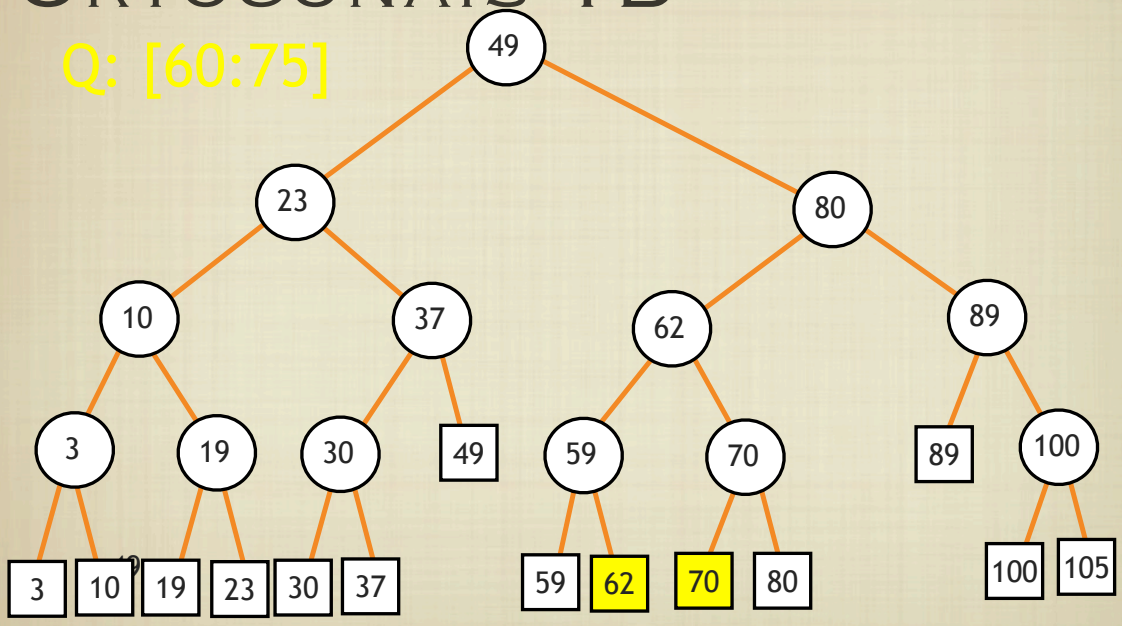



# Buscas Intervalares Ortogonais 1D

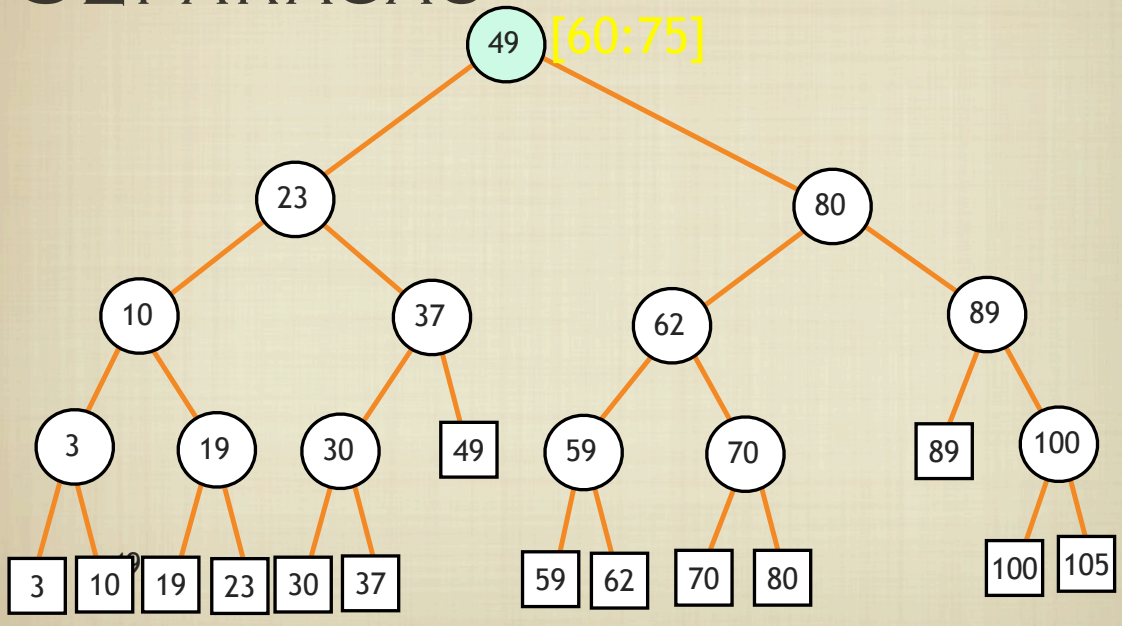
Q: [60:75]



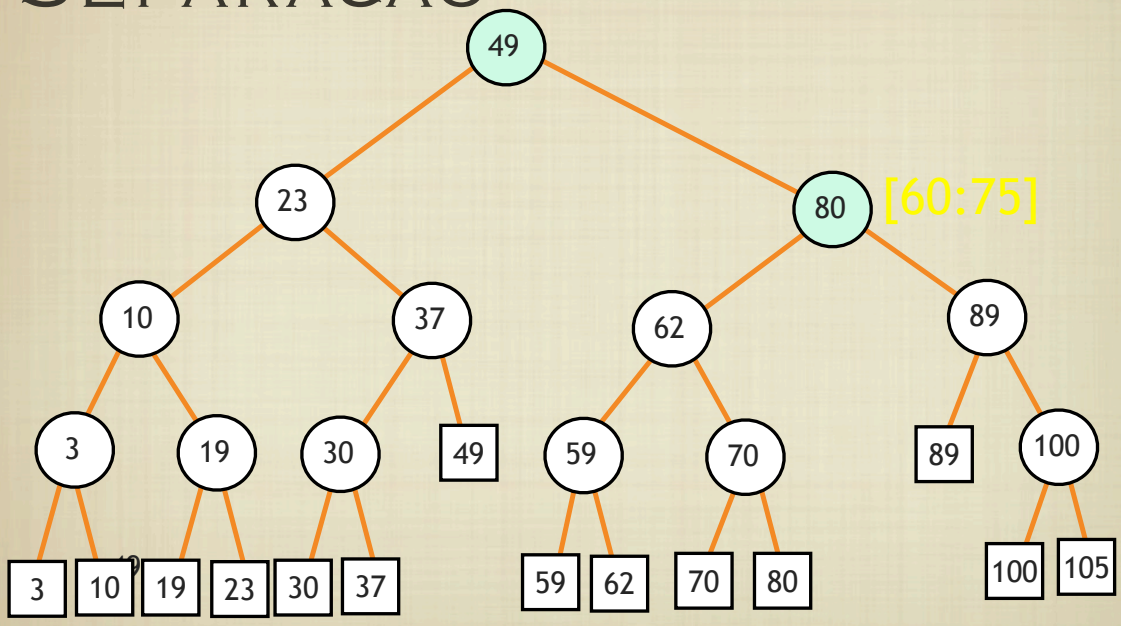
# Buscas Intervalares Ortogonais 1D



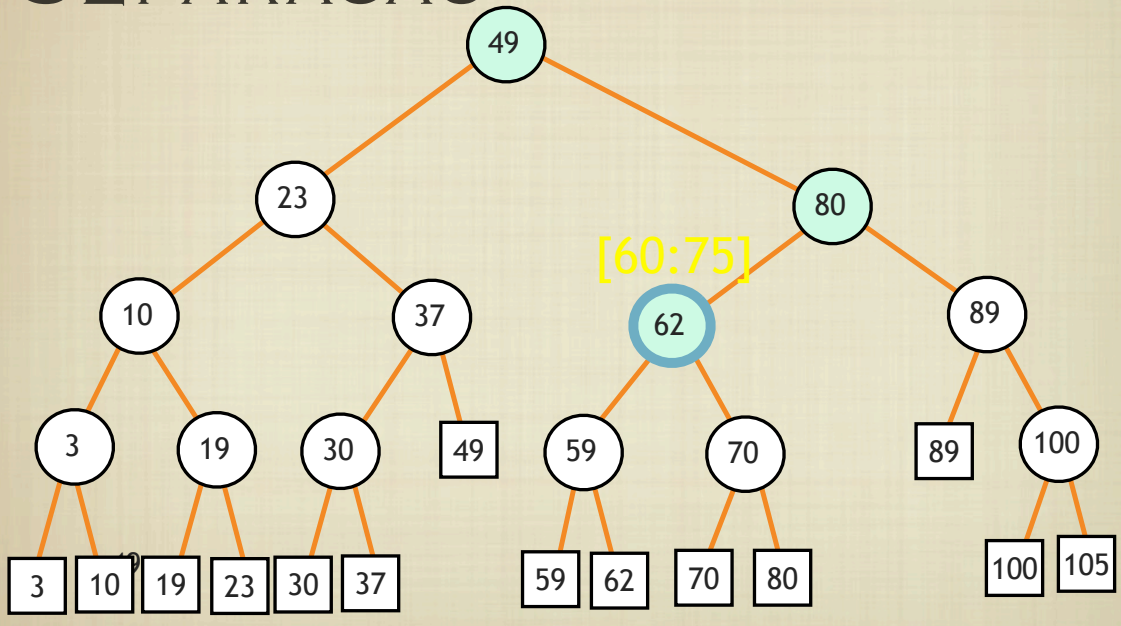

# 1. Busca Nodo de Separacao



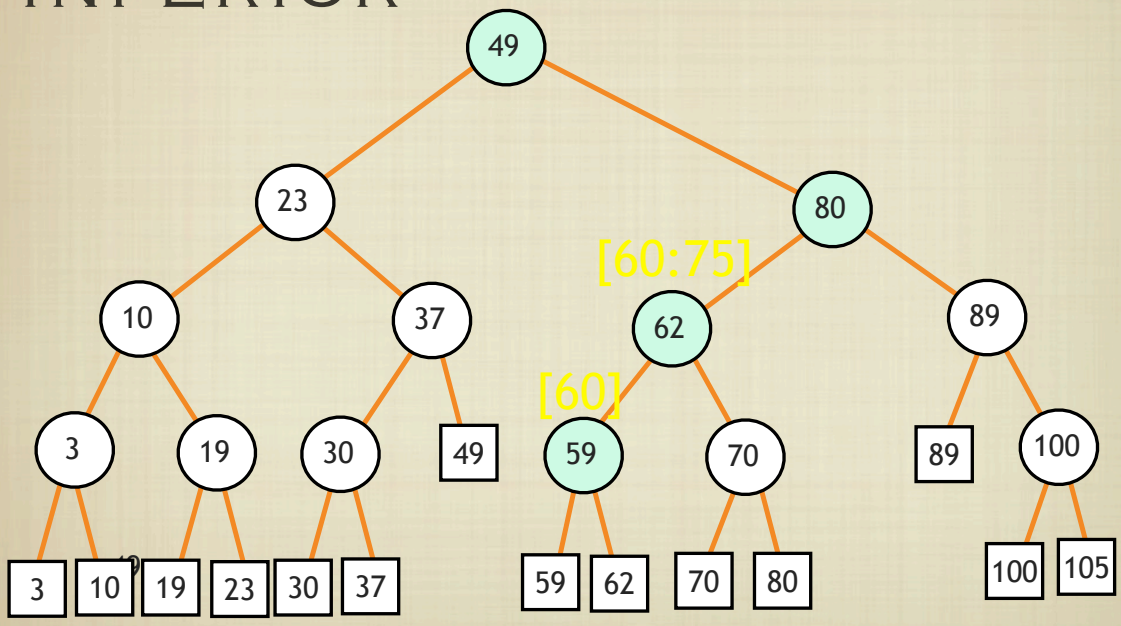
# 1. Busca Nodo de Separacao



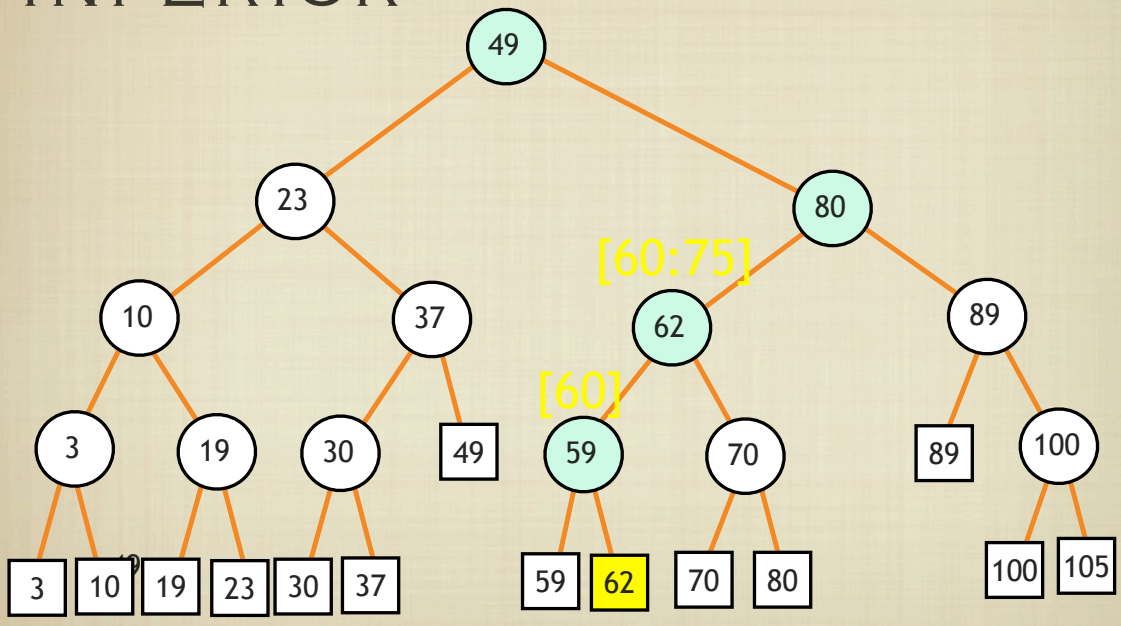

# 1. Busca Nodo de Separacao



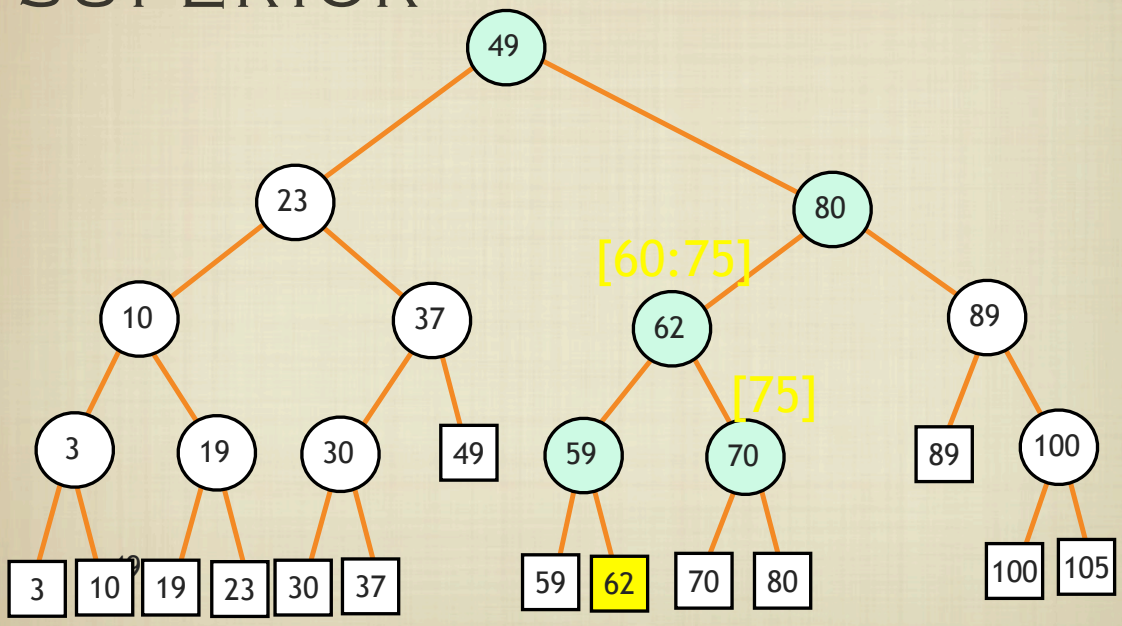
# 2. Encontra limite inferior



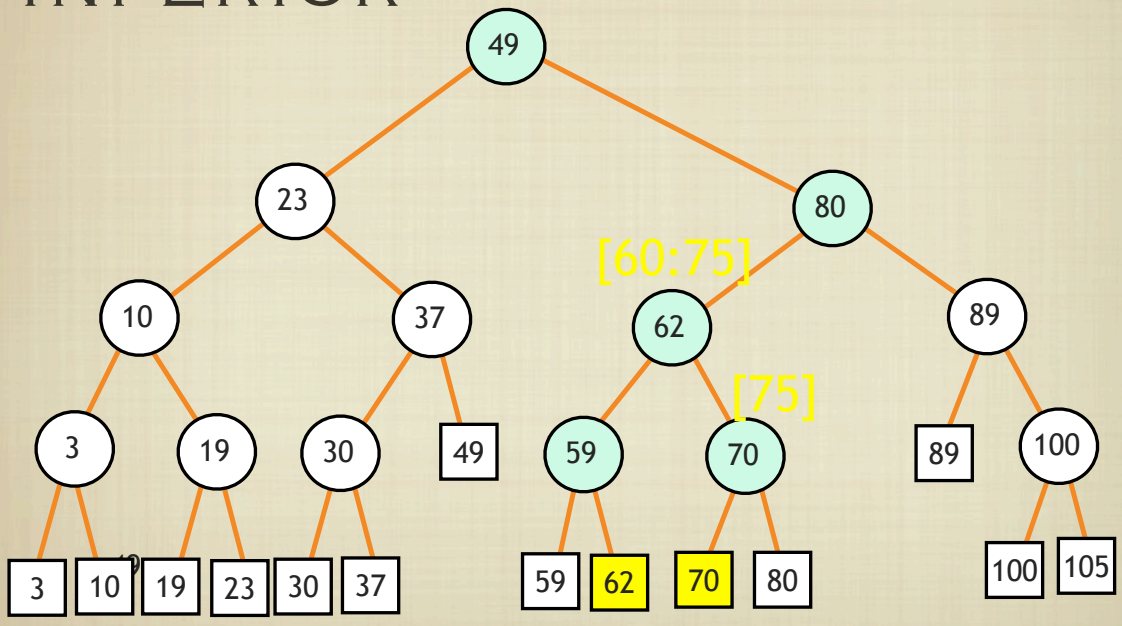

# 2. Encontra limite inferior



# 3. Encontra limite superior

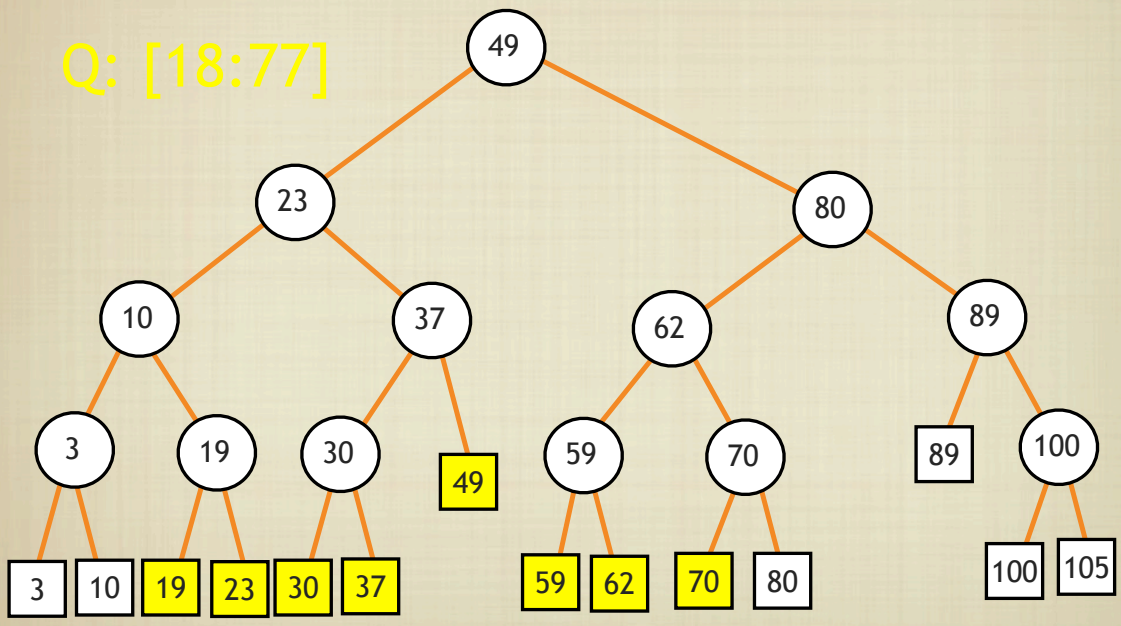



# 3. Encontra limite inferior

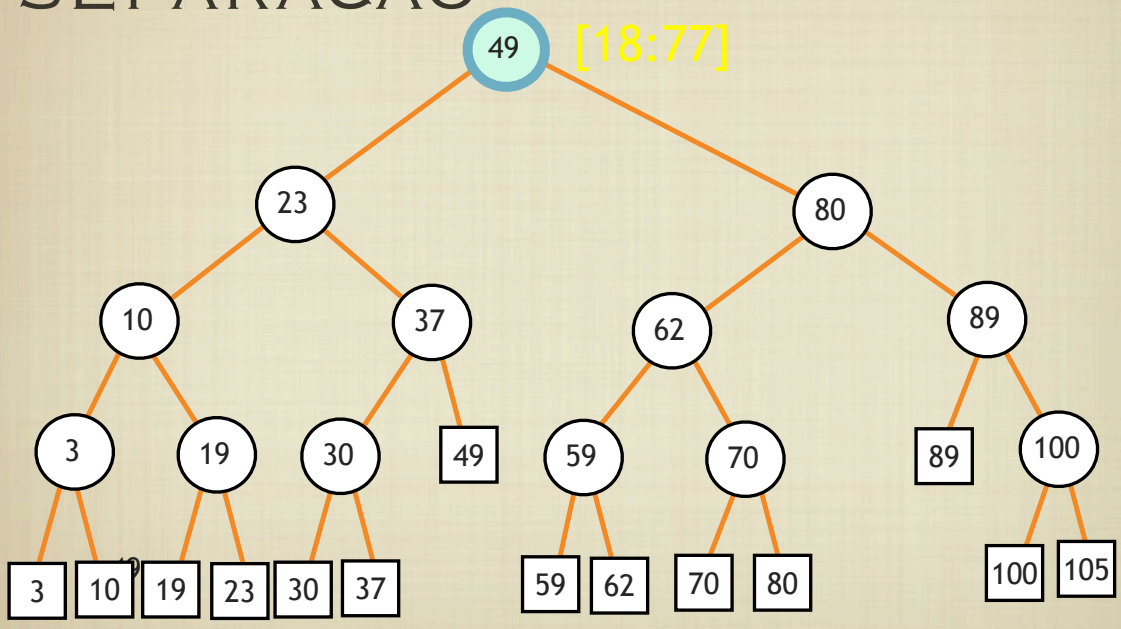


# Outro exemplo

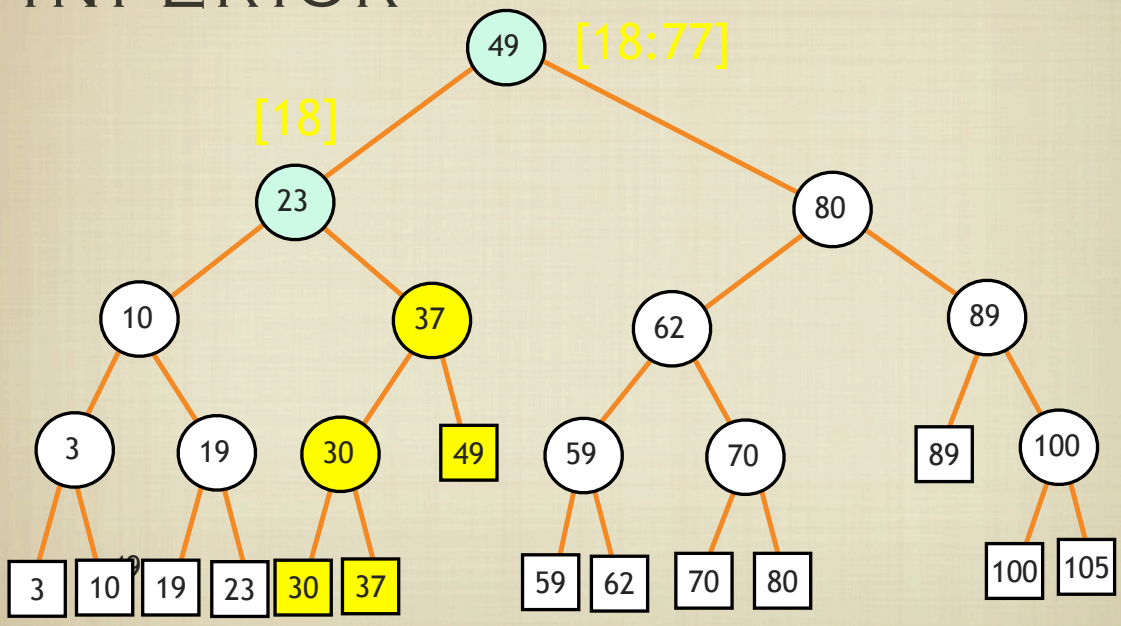
Q: [18:77]



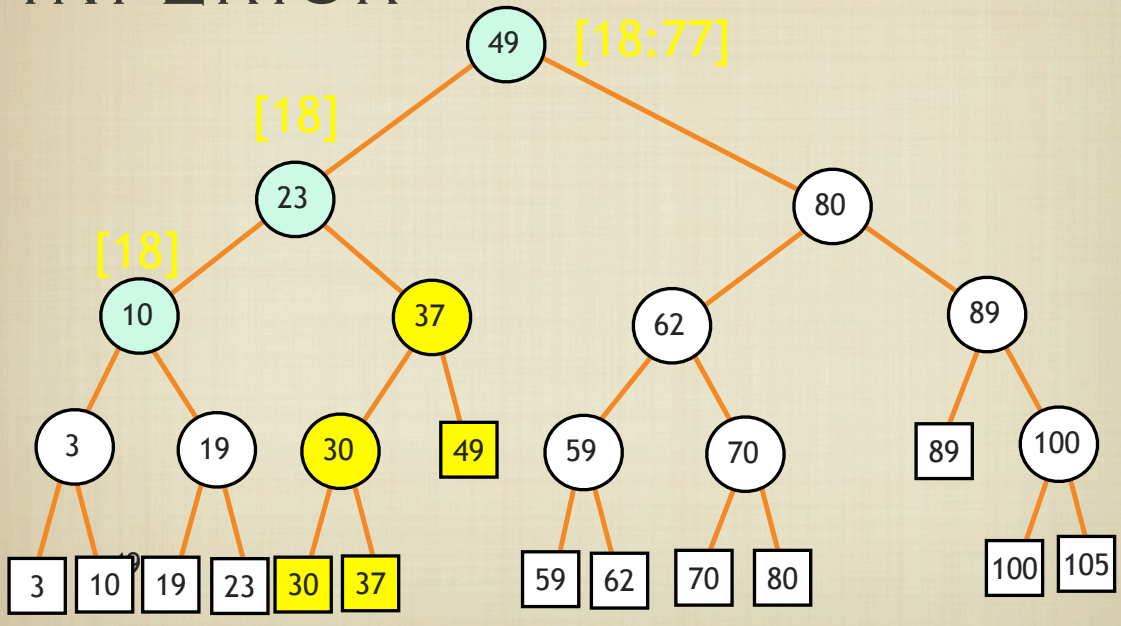

# 1. Busca nodo de separacao



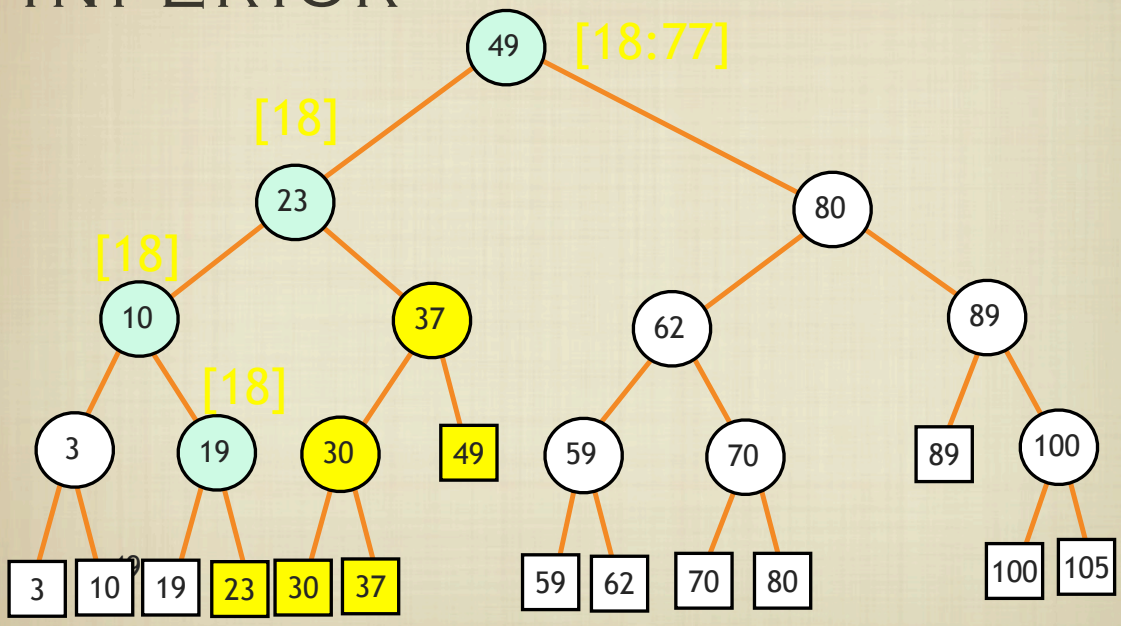
# 2. Encontra limite inferior



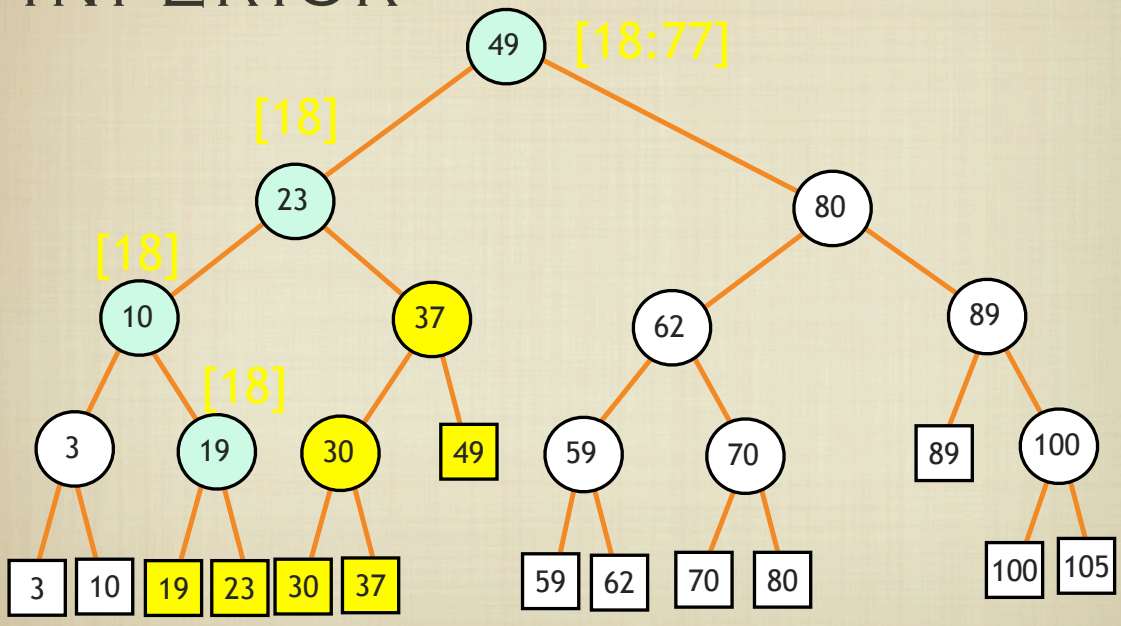

# 2. Encontra limite inferior



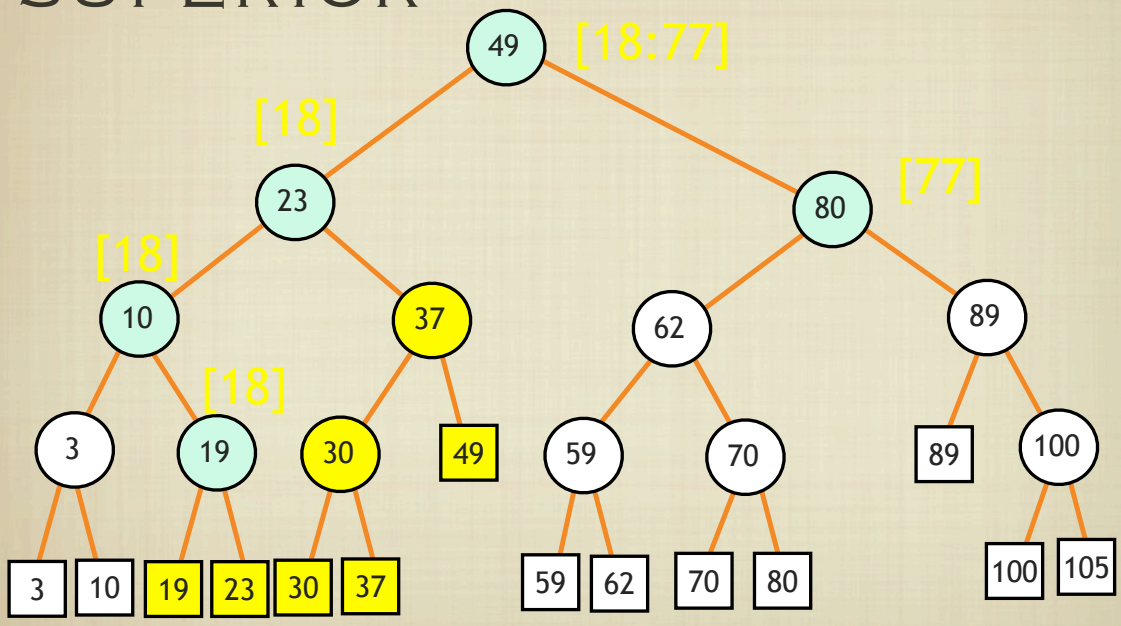
# 2. Encontra limite inferior



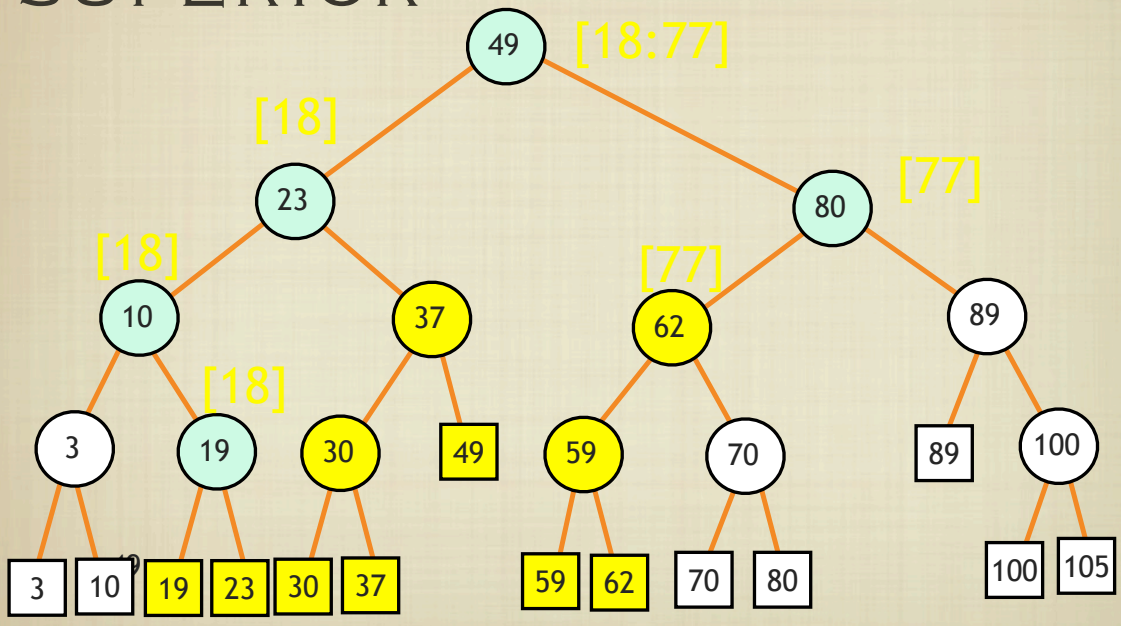

# 2. Encontra limite inferior



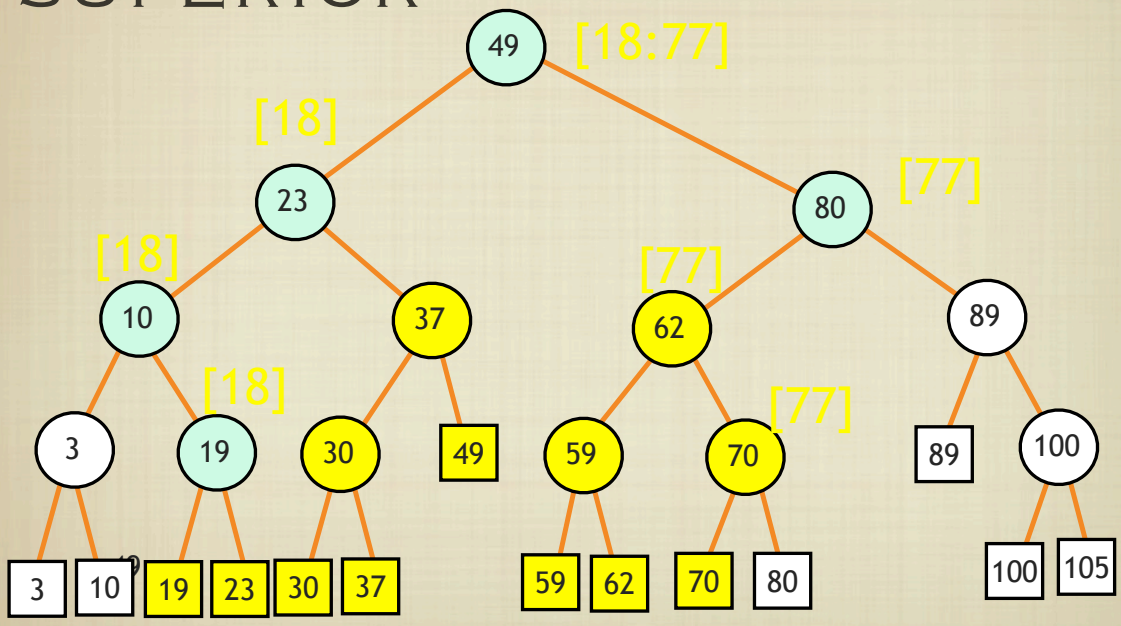
# 3. Encontra limite superior



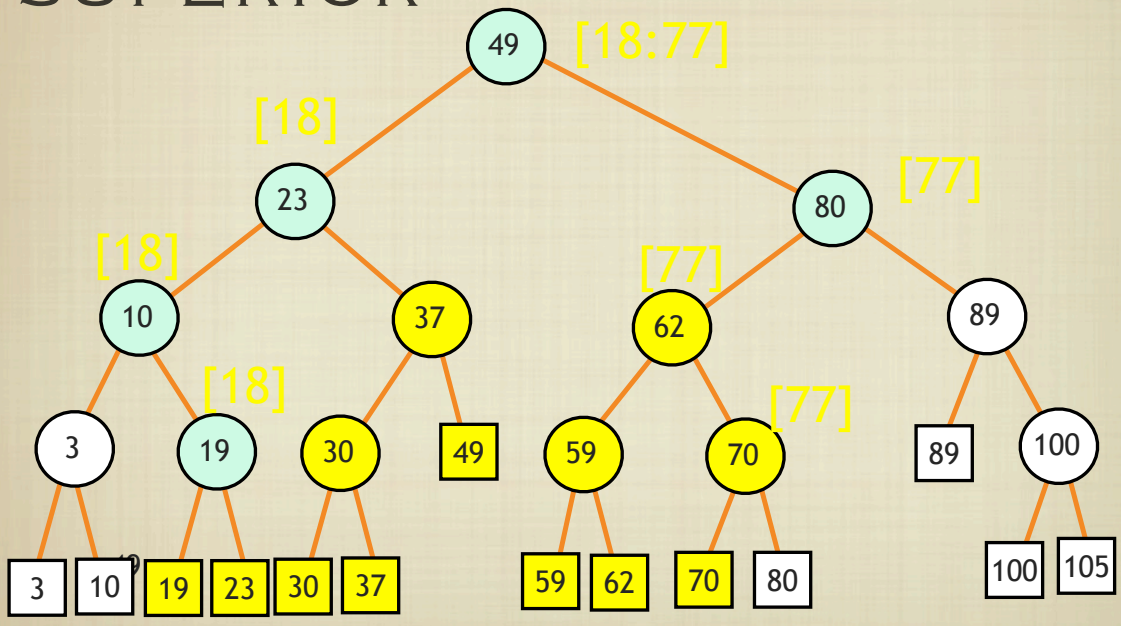

# 3. Encontra limite superior



# 3. Encontra limite superior




# 3. Encontra limite superior



# Algoritmo ENCONTRANODOSEP

```
ENCONTRANODOSEP(T,xmin,xmax)
ENTRADA: T, valores min e max em x
SAIDA: nodo de separacao
1.  v = root(T)
2.  WHILE v nao e' folha e (xmax <= xv OU xmin > xv)
3.  DO IF (xmax <= xv)
4.      THEN v = left(v)
5.      ELSE v = right(v)
6.  RETURN v
```

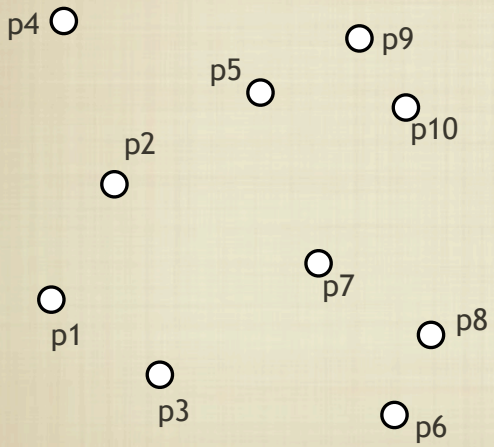
# Algoritmo

```
BUSCARANGETREE1D(T,xmin,xmax)
ENTRADA: T, valores min e max em x
SAIDA: todos nodos no intervalo
1.  vsplit = ENCONTRANODOSEP(T, xmin, xmax)
2.  IF vsplit e' folha
3.  THEN Verificar se ponto na folha deve ser reportado
4.  ELSE //Percorrer arvore ate' xmin, reportar r-subtrees
5.      v = left(vsplit);
6.      WHILE v nao e' folha
7.      DO IF (xmin <= xv)
8.          THEN REPORTA_SUBARVORE(right(v)
9.               v = left(v)
10.         ELSE v = right(v)
11.     // Verificar se ponto na folha deve ser reportado
12.     // Da mesma maneira, percorrer arvore ate' xmax, reportar //
    subtrees a esquerda do caminho, e verificar se ponto na folha deve
    ser reportado
```

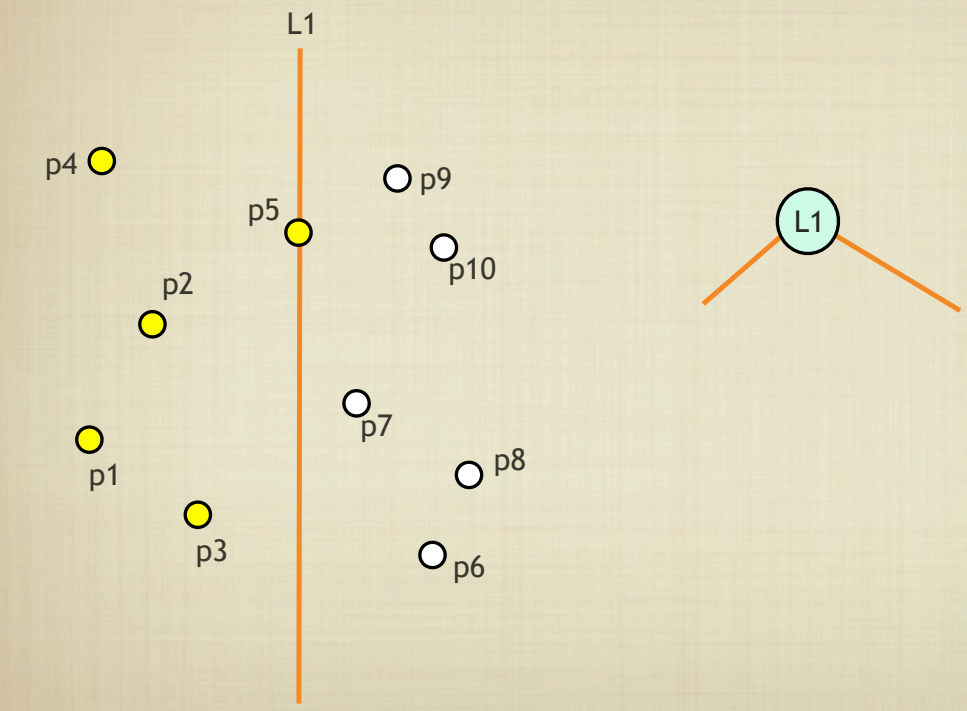
# Performance

- Arvore Binaria Balanceada: O(n) memoria
- Construcao arvore: O(nlogn)
- Consulta ?
  - Pior caso: reportar todos os pontos
  - "output-sensitive": proporcional ao numero de valores reportados (k)
  - O(logn) caminho maximo, O(1) em cada nodo
  - O(k + logn)

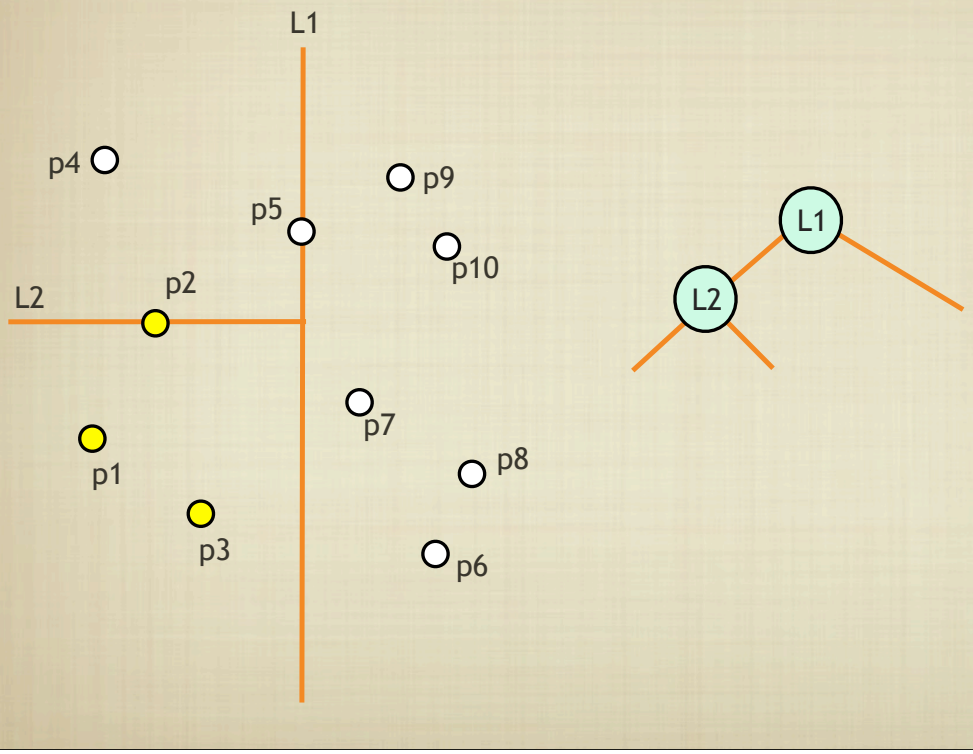
# Kd-trees (2D problem)



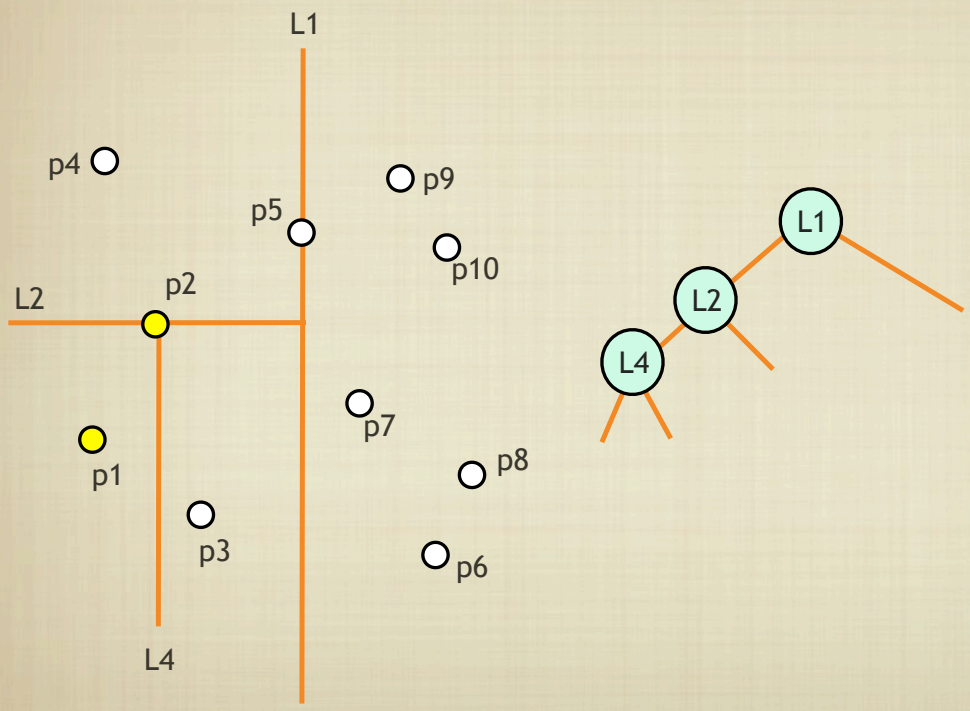
# KD-TREES (2D PROBLEM)



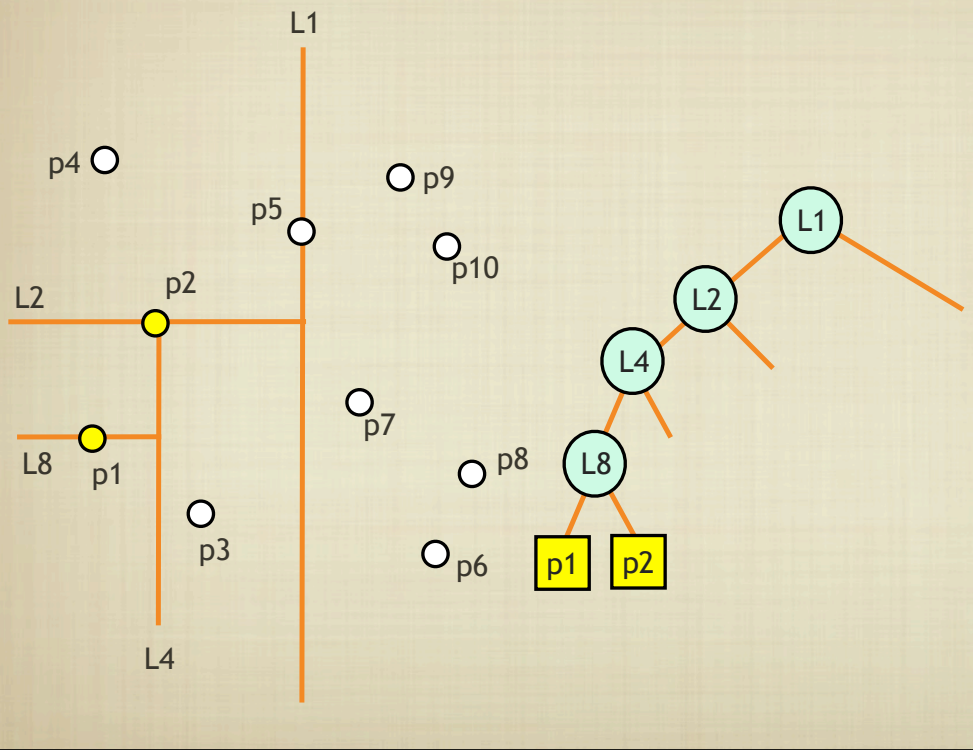

# KD-TREES (2D PROBLEM)



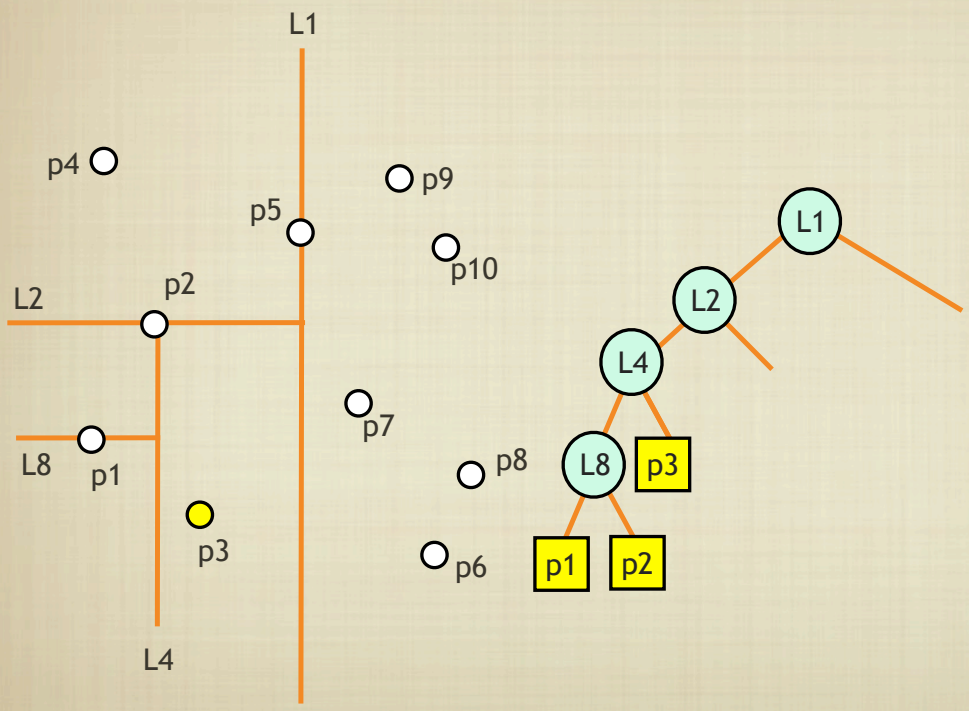
# KD-TREES (2D PROBLEM)



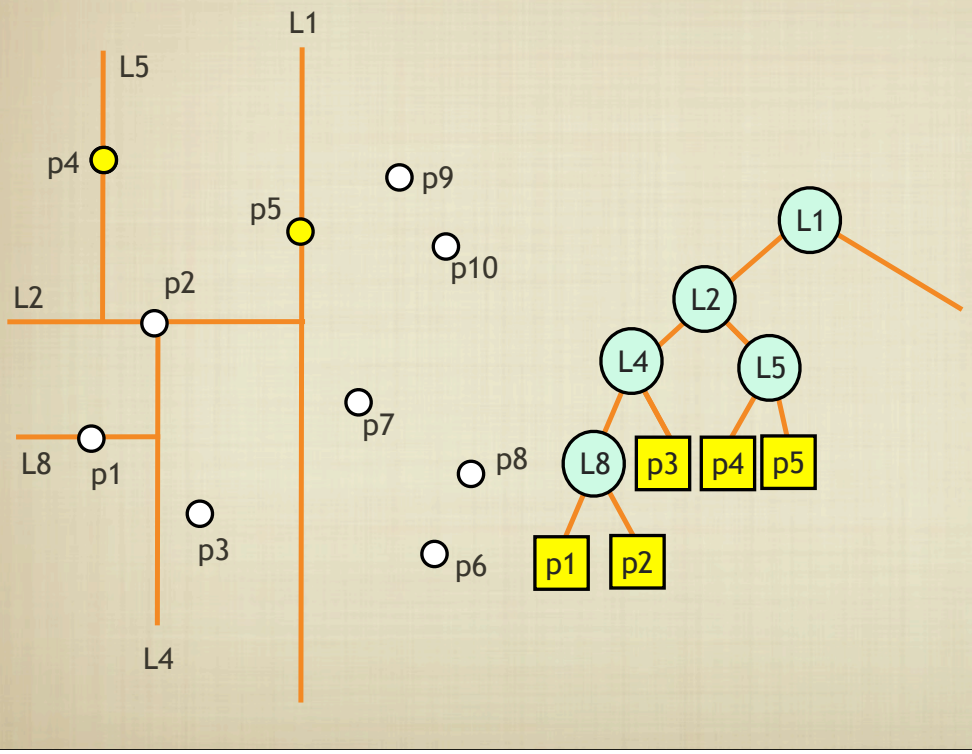

# KD-TREES (2D PROBLEM)



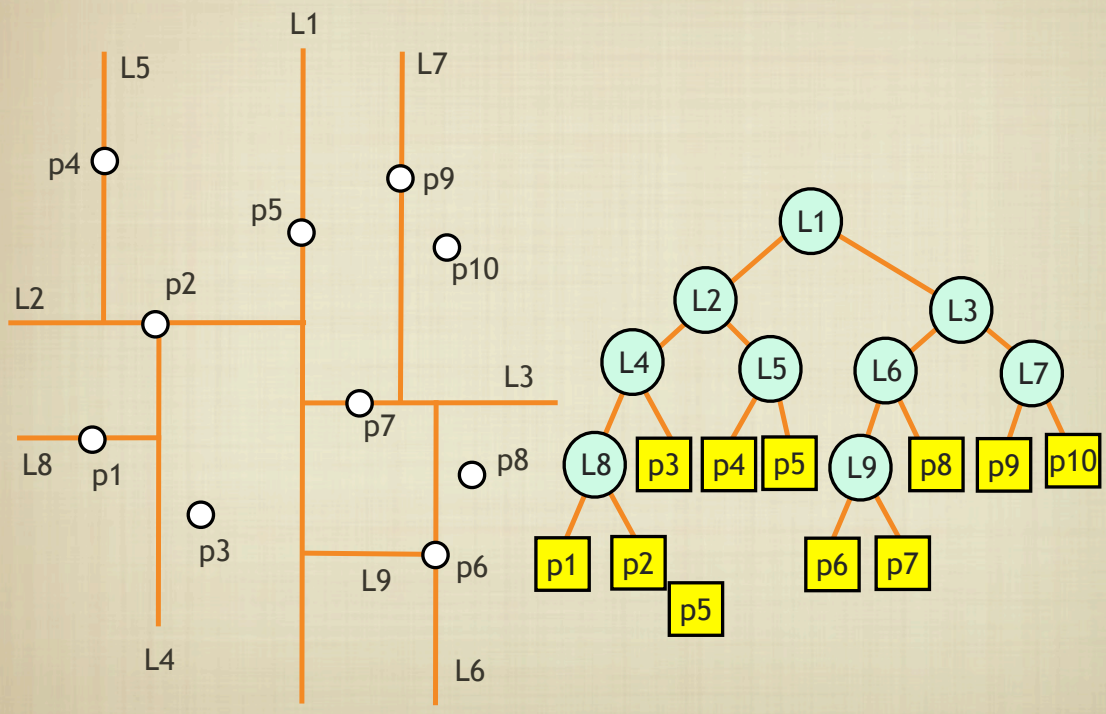
# KD-TREES (2D PROBLEM)




# KD-TREES (2D PROBLEM)



# Kd-trees (2D problem)

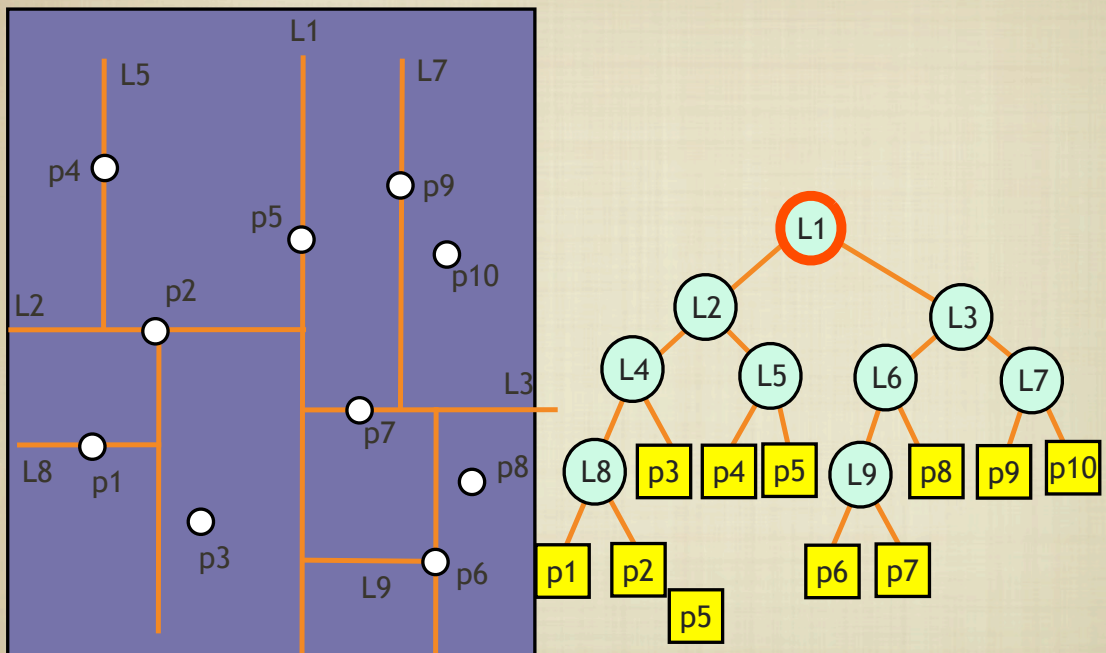



# Algoritmo CONSTROI_KDTREE

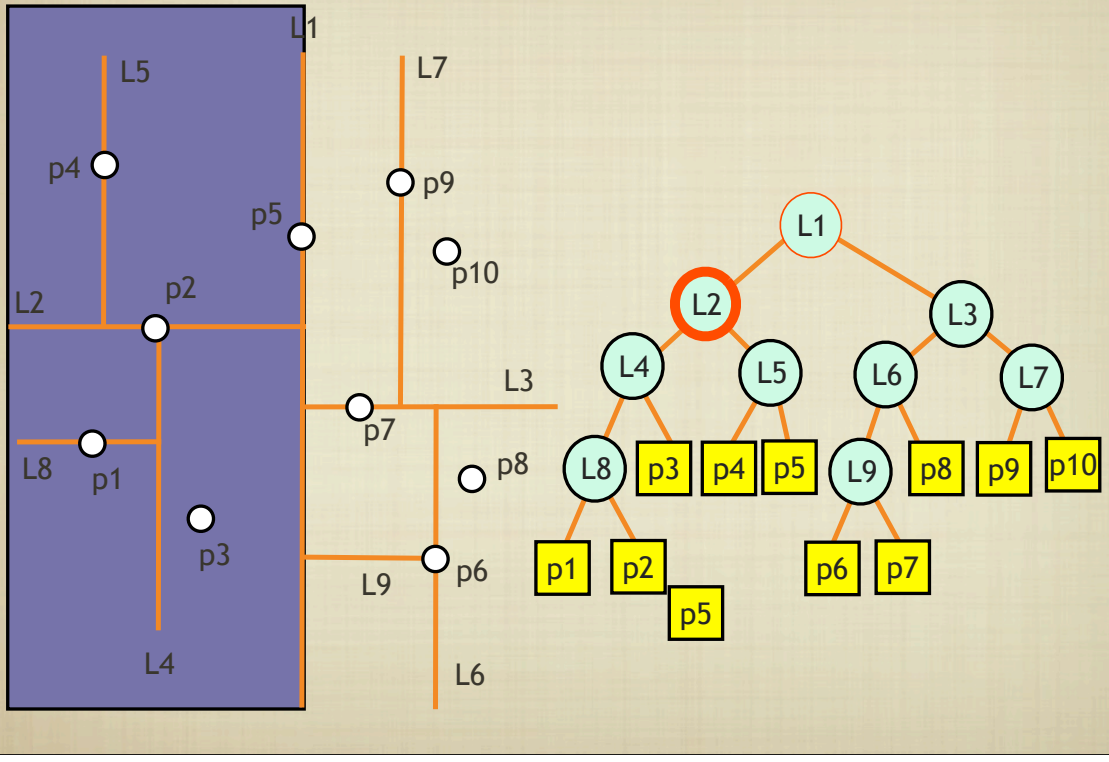
CONSTROI_KDTREE(P, profundidade)
ENTRADA: Conjunto de pontos P, profundidade corrente
SAIDA: todos nodos no intervalo

1. IF P contem um ponto somente
2. THEN RETURN uma folha armazenando este ponto
3. ELSE IF profundidade e' par
4. THEN Particionar P em 2 conjuntos P1(e) e P2(d) com uma linha vertical L passando pela mediana dos pontos em P
5. ELSE Particionar P em 2 conjuntos P1(b) e P2(c) com uma linha vertical L passando pela mediana dos pontos em P
6. vleft = CONSTROI_KDTREE(P1, profundidade+1)
7. vright = CONSTROI_KDTREE(P2, profundidade+1)
8. Criar um nodo v armazenando L, atribuir vleft e vright como as subarvores deste nodoDO IF (xmin <= xv)
9. RETURN v

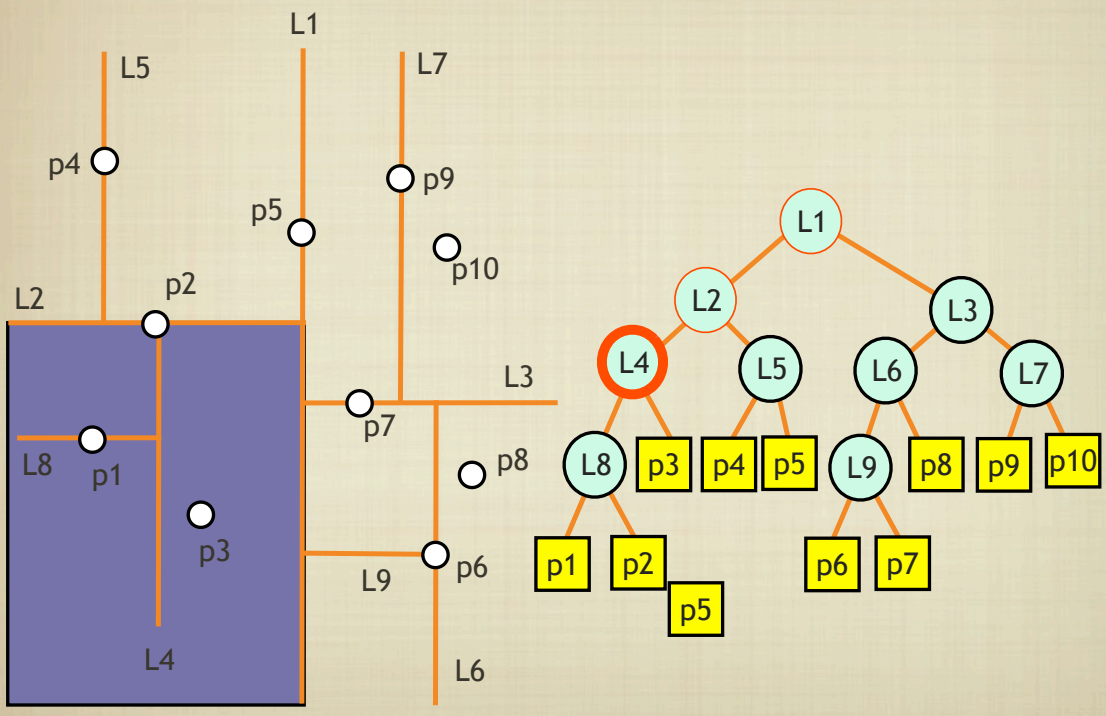
# Correspondencia nodos x regioes



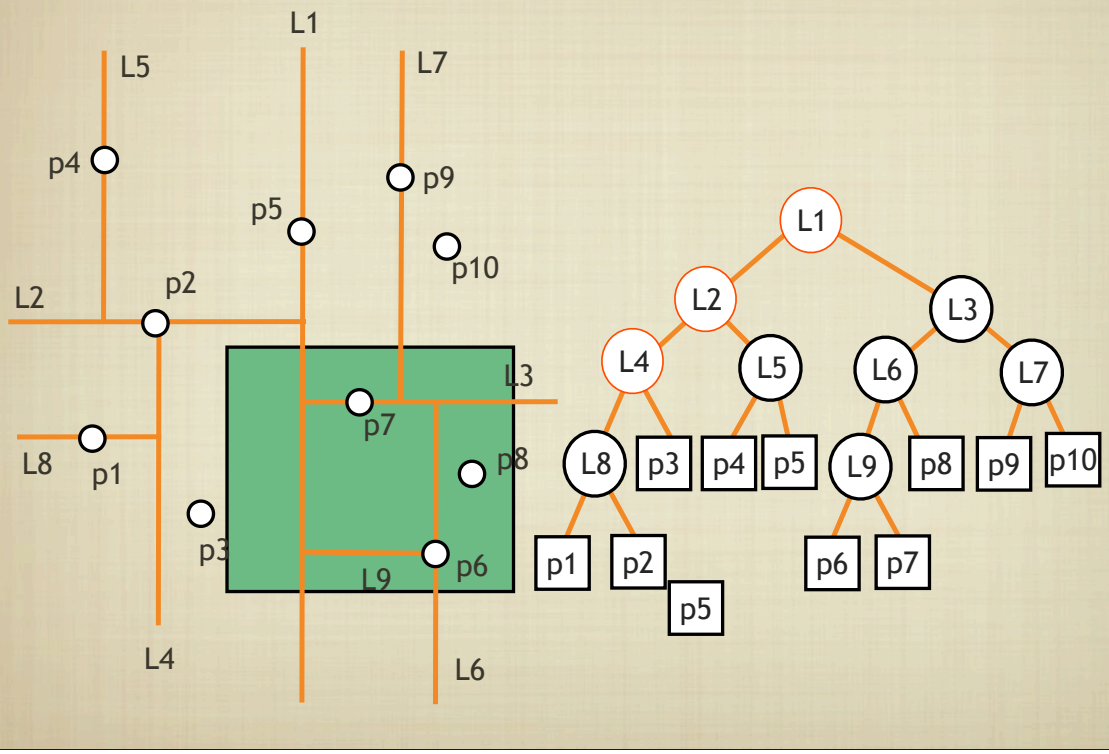

# Correspondencia nodos x regioes



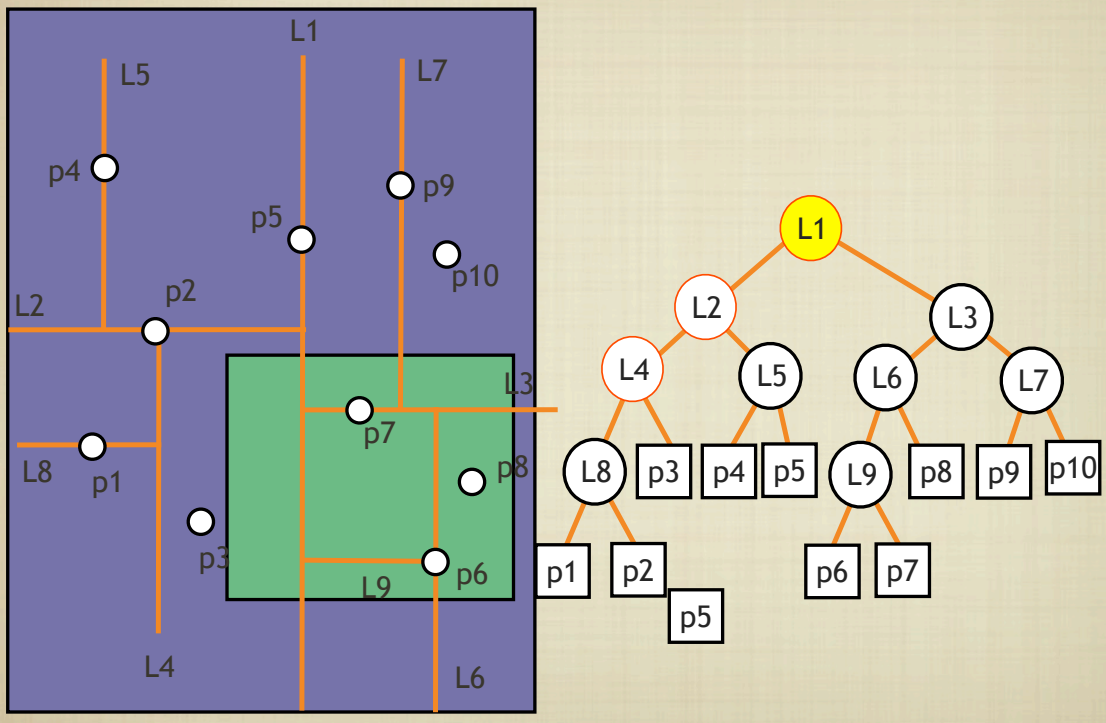
# Correspondencia nodos x regioes



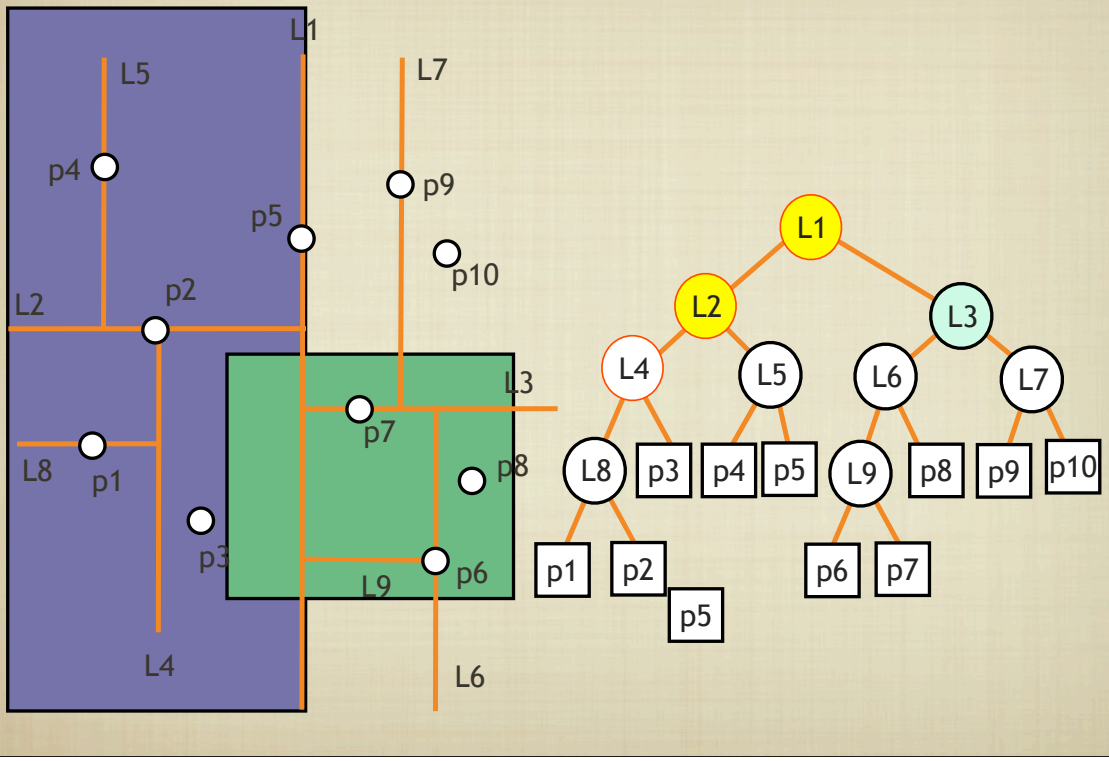

# Busca em KDTrees



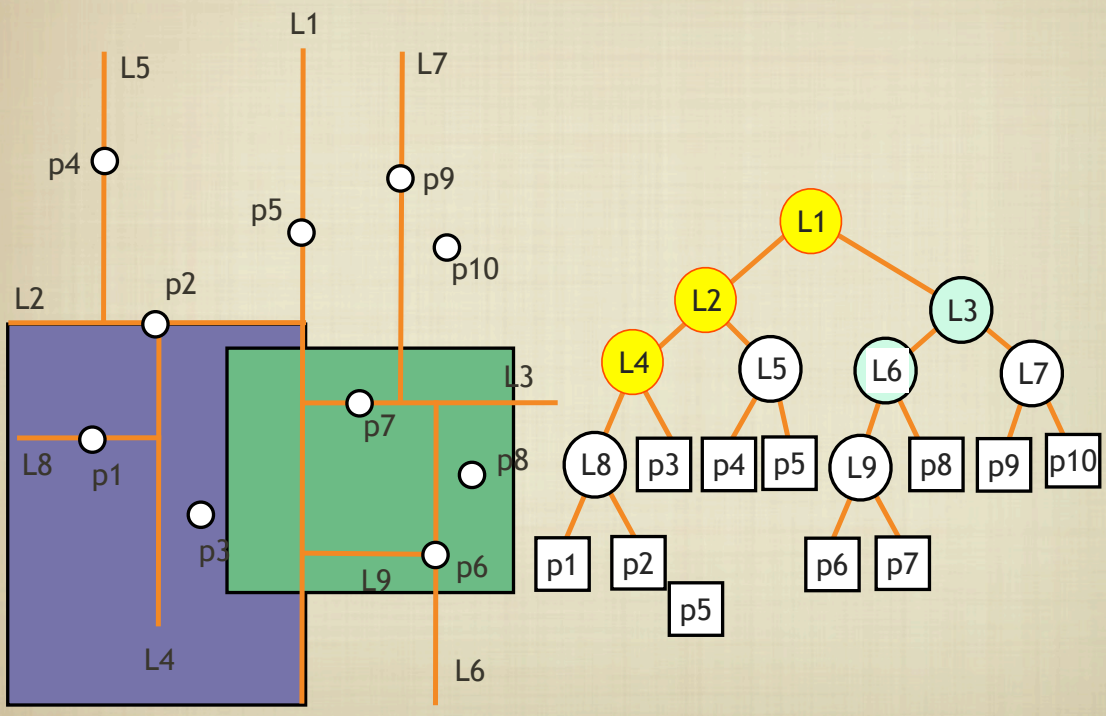
# Busca em KDTrees



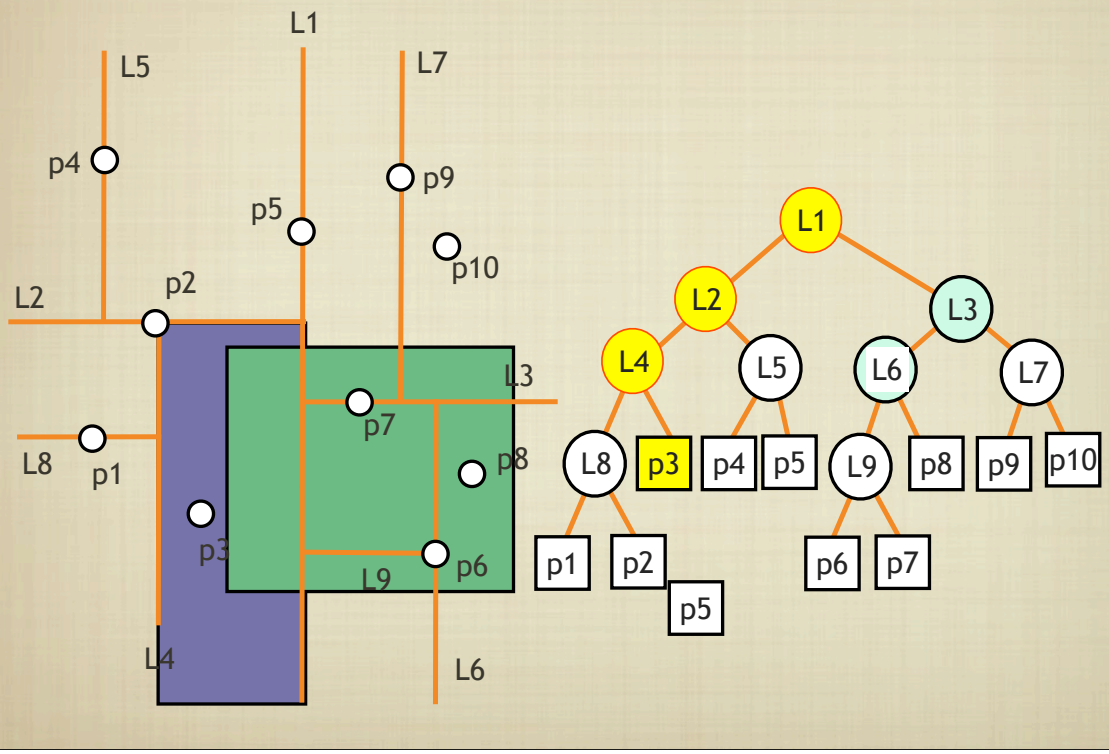

# Busca em KDTrees



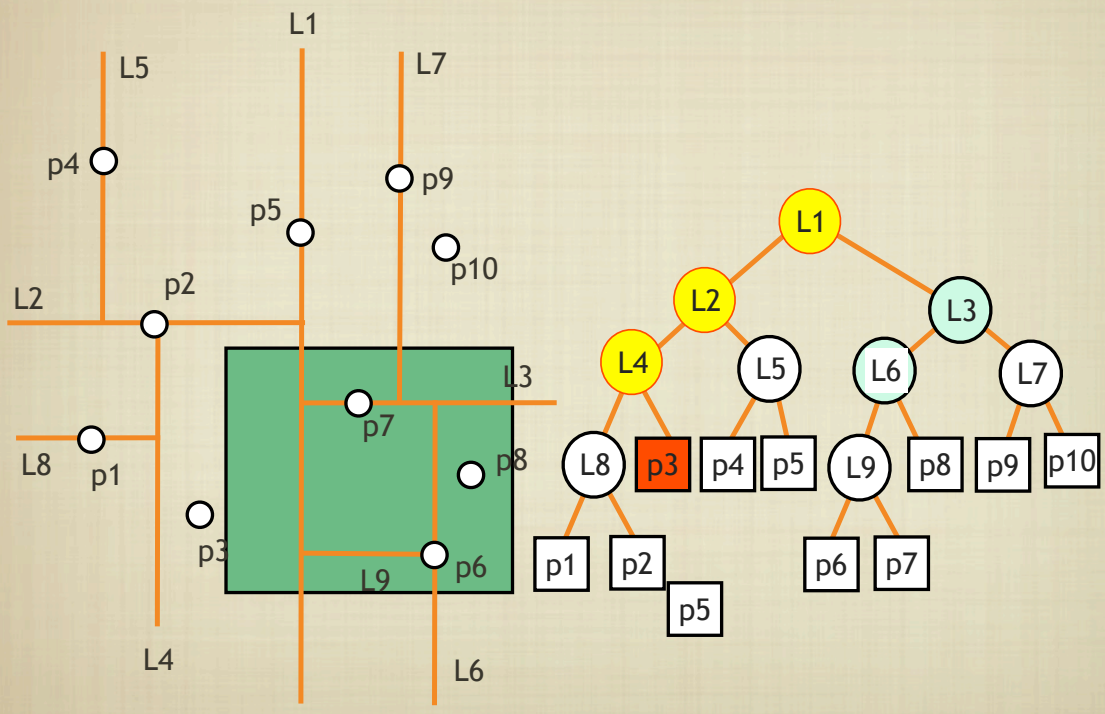
# Busca em KDTrees



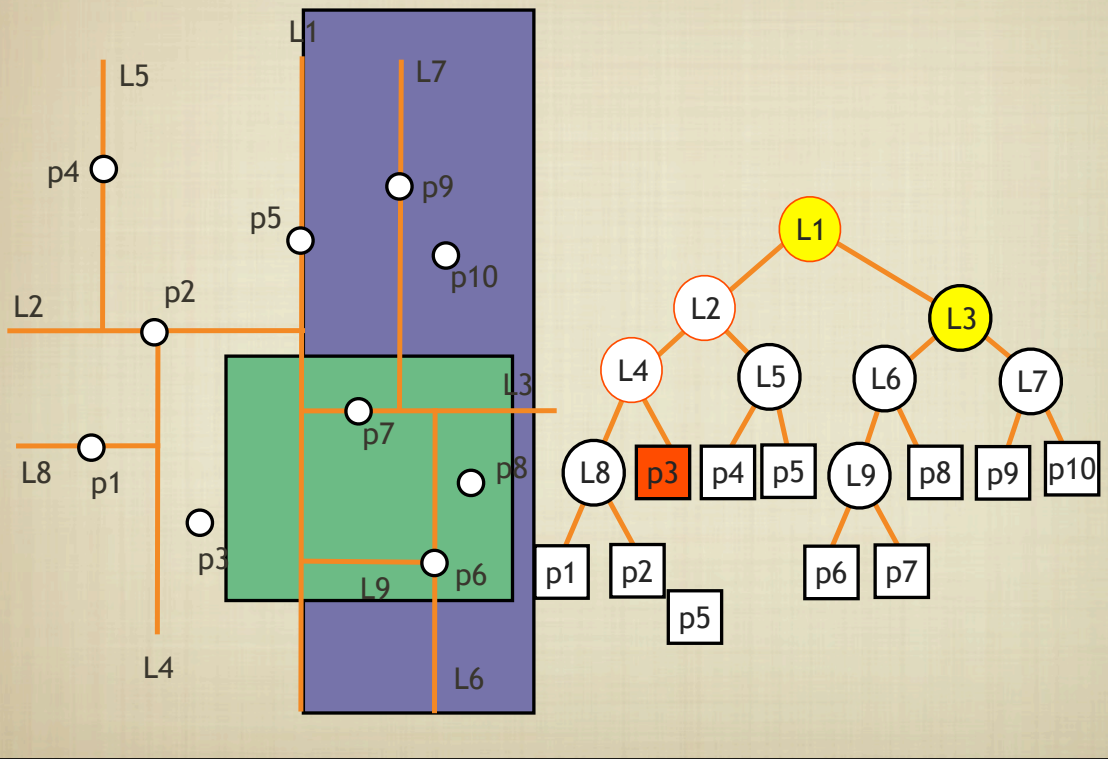

# Busca em KDTrees



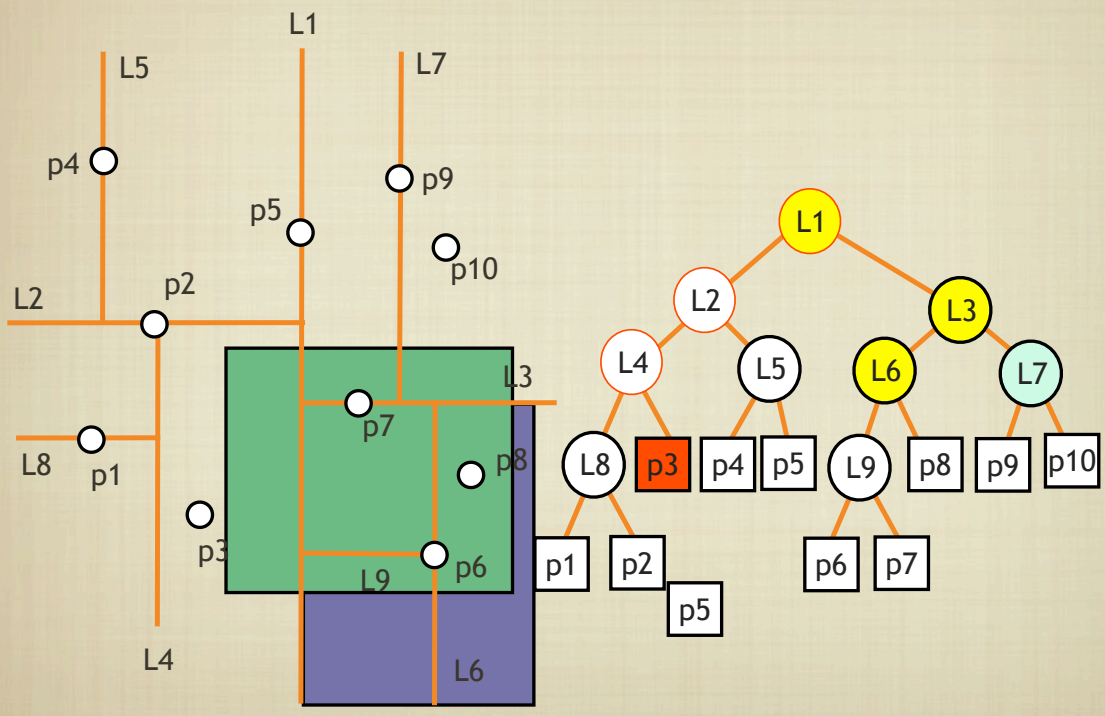
# Busca em KDTrees




# Busca em KDTrees

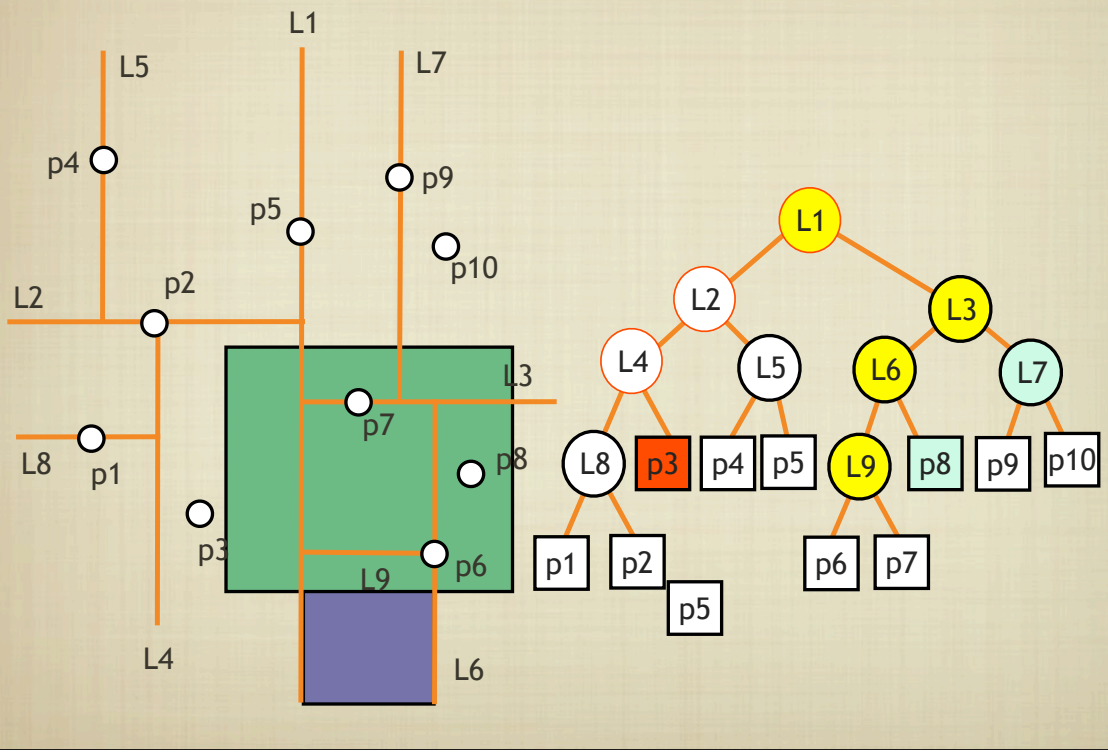


# Busca em KDTrees

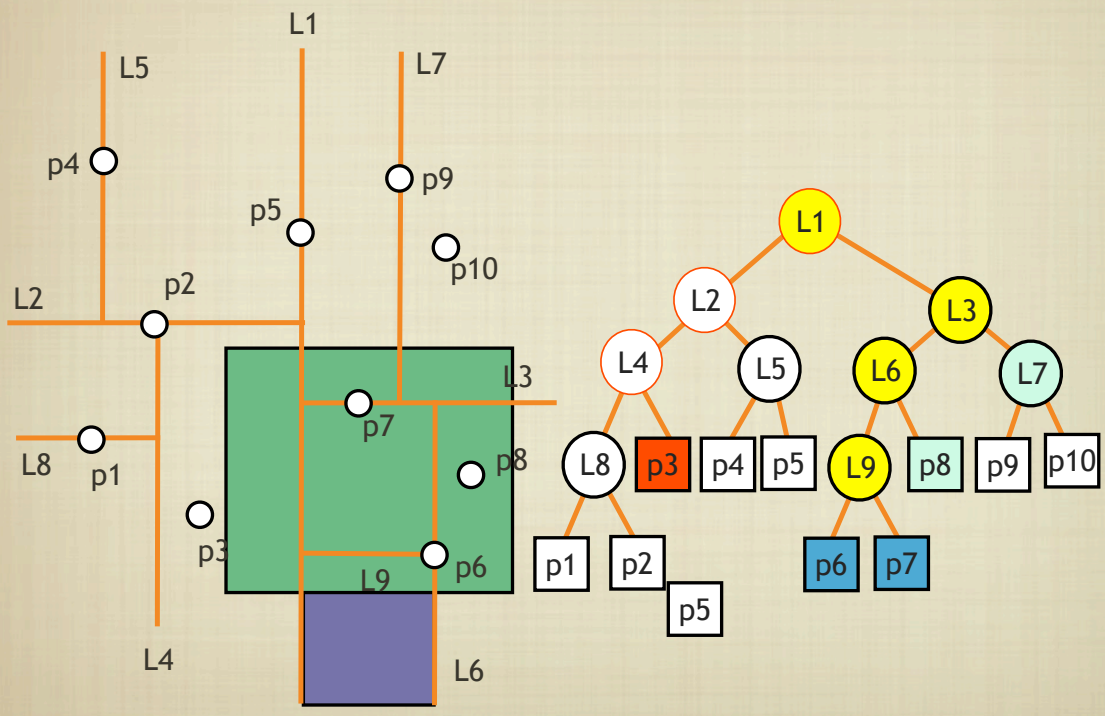




# Busca em KDTrees

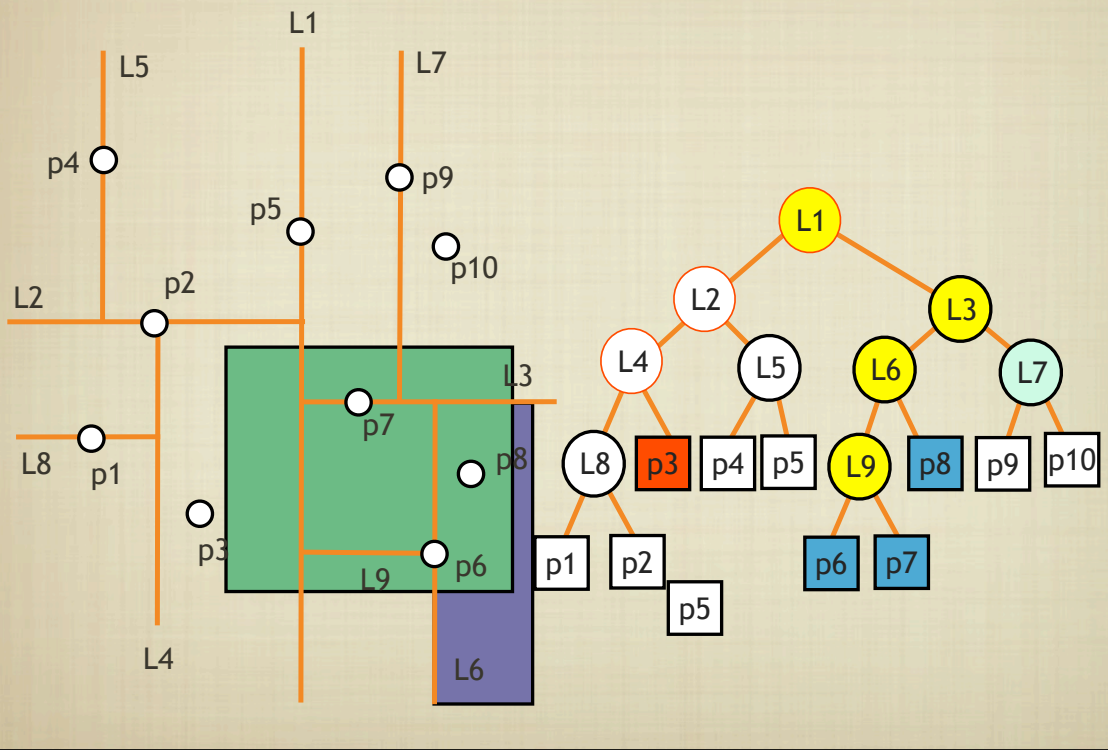



# Busca em KDTrees

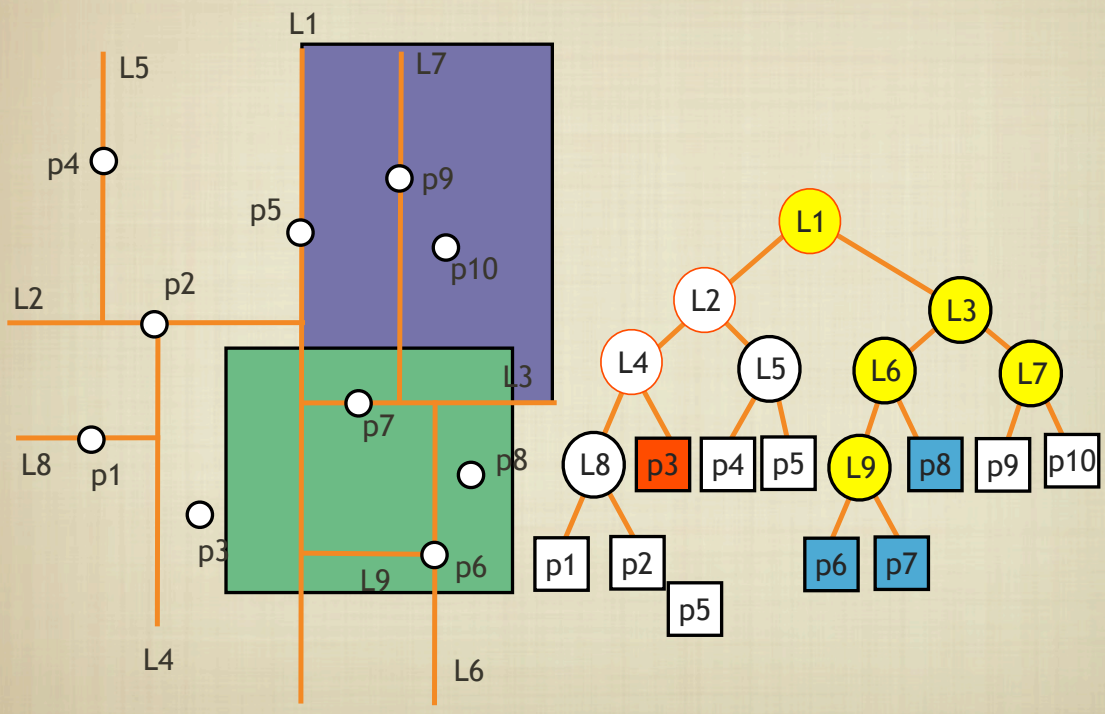




# Busca em KDTrees

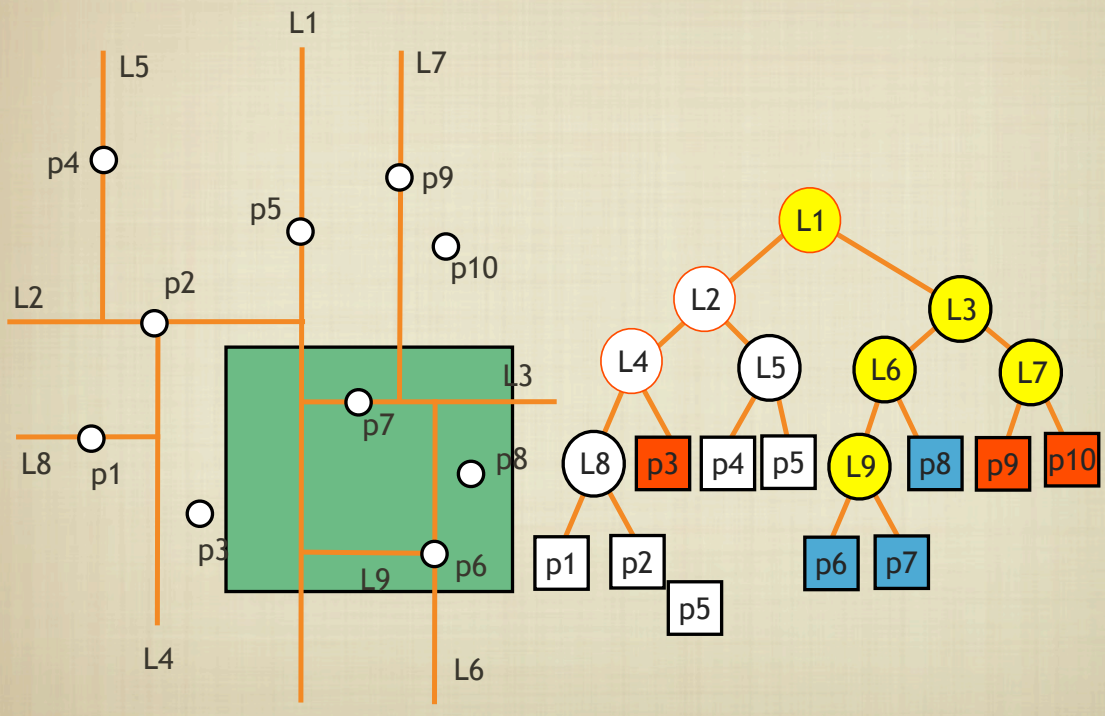



# Busca em KDTrees





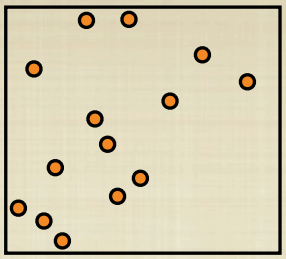
# Busca em KDTrees

# Algoritmo BUSCA_KDTREE

```
BUSCA_KDTREE(v, R)
ENTRADA: Raiz da kdtree e um intervalo 2D R
SAIDA: todas folhas contendo ponto no intervalo
1.  IF v e' uma folha
2.  THEN reportar todos pontos em v que estao dentro de R
3.  ELSE IF REGIAO(left(v)) esta contida em R
4.       THEN REPORTA_SUBARVORE(left(v))
5.       ELSE IF REGIAO(left(v)) intersecta R
6.            THEN BUSCA_KDTREE(left(v), R)
7.       IF REGIAO(right(v)) esta contida em R
8.       THEN REPORTA_SUBARVORE(right(v))
9.       ELSE IF REGIAO(right(v)) intersecta R
10.           THEN BUSCA_KDTREE(right(v), R)
```

# Exemplo



| 2 | 5 | 7 | 8 | 12 | 15 | 17 | 21 | 33 | 41 | 52 | 58 | 67 | 93 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 19 | 80 | 10 | 37 | 3 | 99 | 62 | 49 | 30 | 95 | 23 | 69 | 89 | 70 |

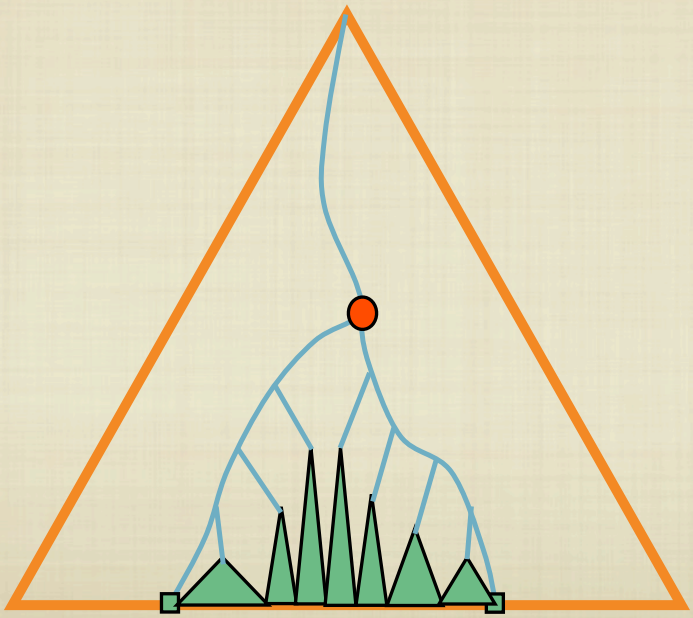
# Performance

- Arvore Binaria: O(n) memoria
- Construcao arvore: O(nlogn)
- Consulta ?
  - O(k + sqroot(n))
  - Q(n) = [O(1) se n = 1, 2 + 2Q(n/4)]

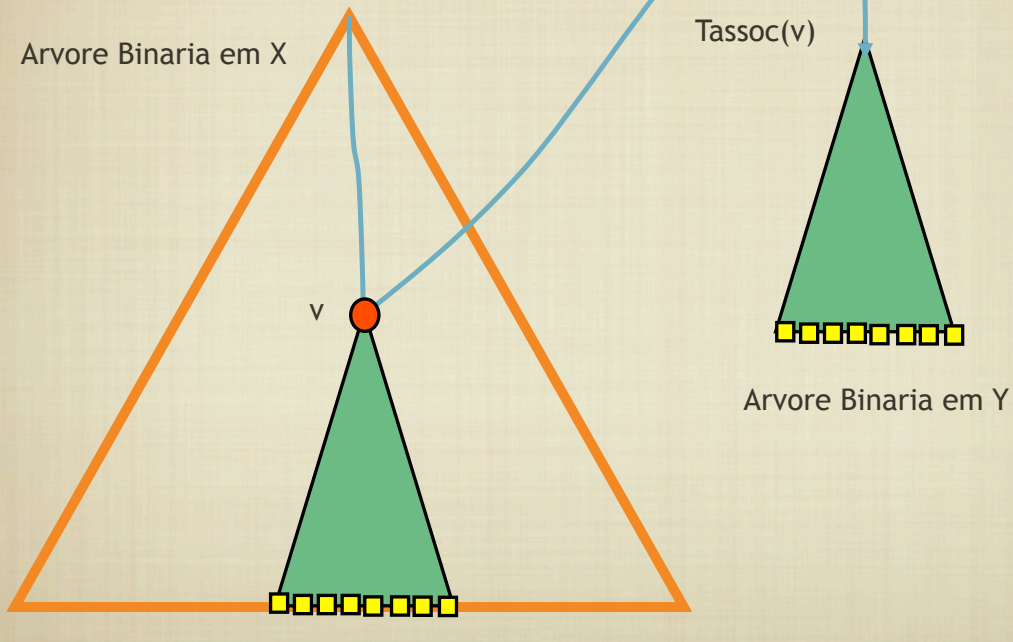
# Range Trees

- **OBS: Consulta 2D corresponde a 2 consultas 1D (em x, e depois em y)**
- **Estrategia [xmin:xmax]x[ymin:ymax]**
  - Construir uma arvore binaria TX em X
  - Encontrar os pontos cujas coordenadas x estejam entre xmin e xmax em TX
    - Usar algoritmo descrito primeiro na aula
  - Construir uma arvore binaria TY com os pontos encontrados em Y
  - Encontrar os pontos cujas coordenadas y estejam entre ymin e ymax

# Range Trees



# Range Trees



# Algoritmo CONSTROI_RANGETREE_2D

CONSTROI_RANGETREE_2D(P)
ENTRADA: Conjunto de ponto P no plano
SAIDA: Raiz de uma range tree 2D

1. Constroi a arvore associada: Criar uma arvore binaria Tassoc no conjunto de coordenadas y dos pontos em P. Armazenar nas folhas nao a coordenada y, mas os pontos em si
2. IF P contem um ponto
3. THEN Criar uma folha v contendo a coord. x e a arvore associada
4. ELSE Particionar P em dois conjuntos P1(l) e P2(l) pela mediana da coordenada x.
5. vleft = CONSTROI_RANGETREE_2D(P1)
6. vright = CONSTROI_RANGETREE_2D(P2)
7. Criar um nodo v contendo a coordenada x da mediana, as folhas esquerda e direita iguais a vleft e vright, e a arvore associada
8. RETURN v

# Algoritmo

BUSCARANGETREE2D(T,[xmin,xmax]x[ymin,ymax])
ENTRADA: Range tree 2D, intervalo 2D, SAIDA: Pontos na range tree dentro do intervalo

1. vsplit = ENCONTRANODOSEP(T, xmin, xmax)
2. IF vsplit e' folha
3. THEN Verificar se ponto na folha deve ser reportado
4. ELSE //Percorrer arvore ate' xmin, chamar BUSCARANGETREE1D // com as r-subtrees
5. v = left(vsplit);
6. WHILE v nao e' folha
7. DO IF (xmin <= xv)
8. THEN BUSCARANGETREE1D(TASSOC(right(v)),[ymin,ymax])
9. v = left(v)
10. ELSE v = right(v)
11. // Verificar se ponto na folha deve ser reportado
12. // Percorrer arvore ate' xmax, chamar BUSCARANGETREE1D , e verificar se ponto na folha deve ser reportado

# Performance

- Arvore Binaria * logn: O(nlogn) memoria
- Construcao range tree: O(nlogn)
  - Requer ordenacao dos pontos em y e em x
- Consulta ?
  - $O(k + Log^2(n))$

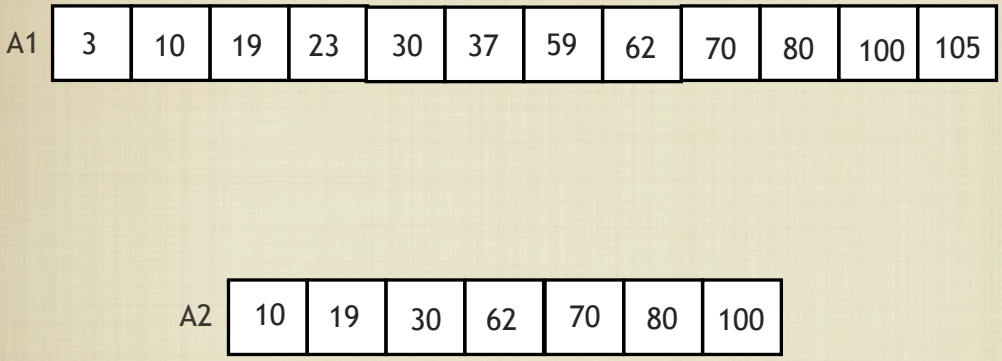
# Cascadeamento Fracionário

| A1 | 3 | 10 | 19 | 23 | 30 | 37 | 59 | 62 | 70 | 80 | 100 | 105 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

| A2 | 10 | 19 | 30 | 62 | 70 | 80 | 100 |
|---|---|---|---|---|---|---|---|

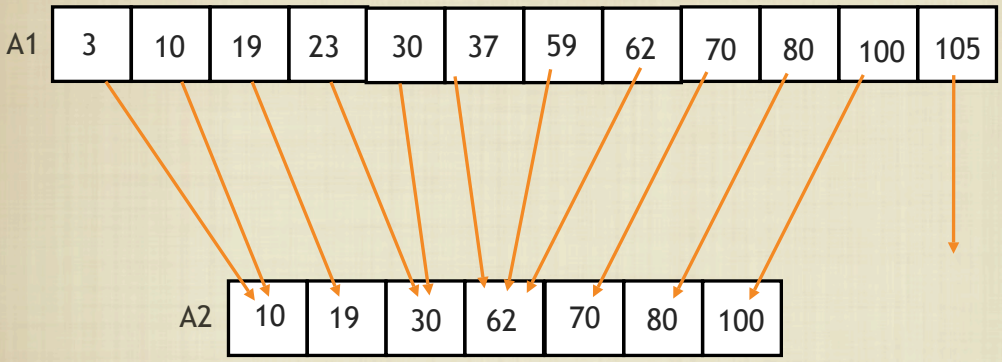
- A2 e' um subconjunto de A1
- 2 buscas

# CASCADEAMENTO FRACIONÁRIO




Colocar um pointer de cada entrada a1 em A1 para a entrada em A2 com a menor chave maior ou igual a a1, ponteiro nulo caso contrario

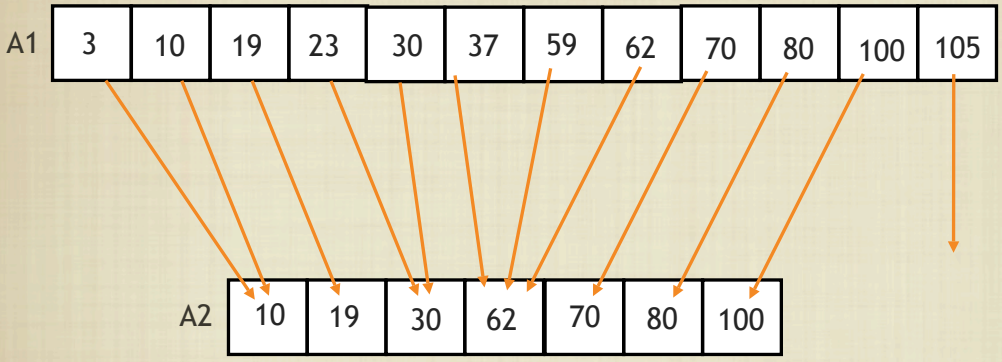

# CASCADEAMENTO FRACIONÁRIO



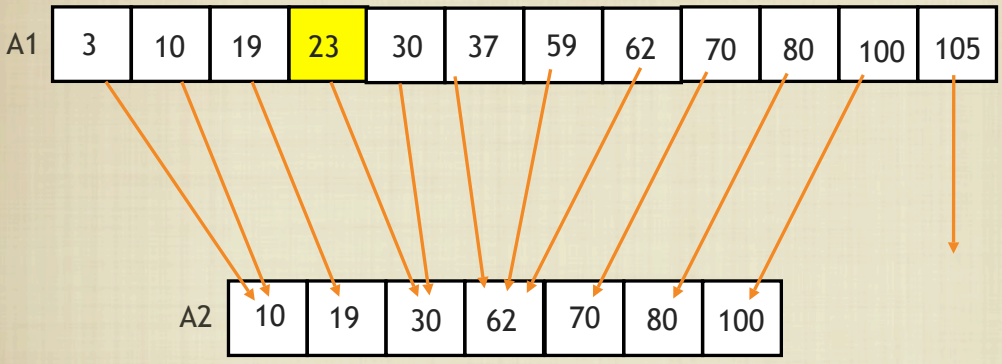
Colocar um pointer de cada entrada a1 em A1 para a entrada em A2 com a menor chave maior ou igual a a1, ponteiro nulo caso contrario

# CASCADEAMENTO FRACIONÁRIO



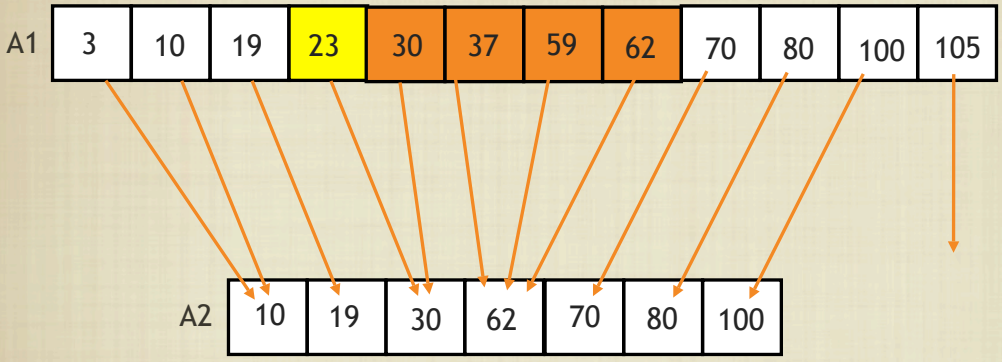

Exemplo: [20:65]

# CASCADEAMENTO FRACIONÁRIO



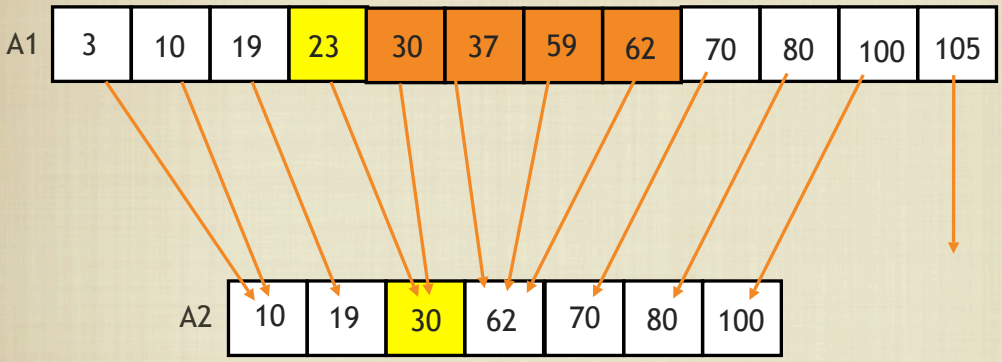

Exemplo: [20:65]

# CASCADEAMENTO FRACIONÁRIO




Exemplo: [20:65]

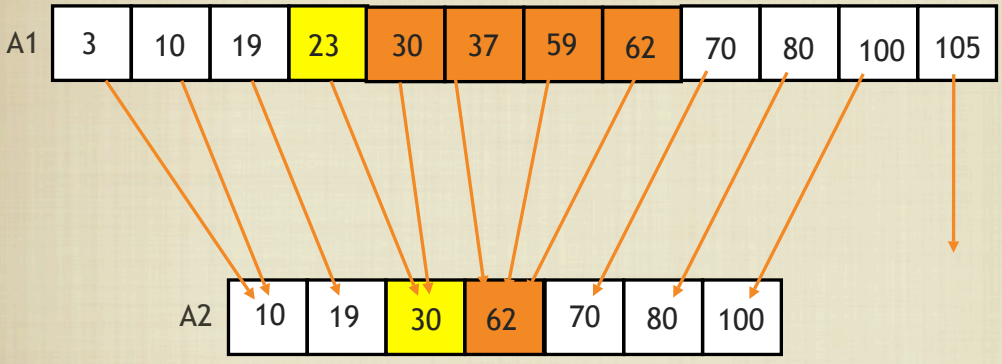

# CASCADEAMENTO FRACIONÁRIO

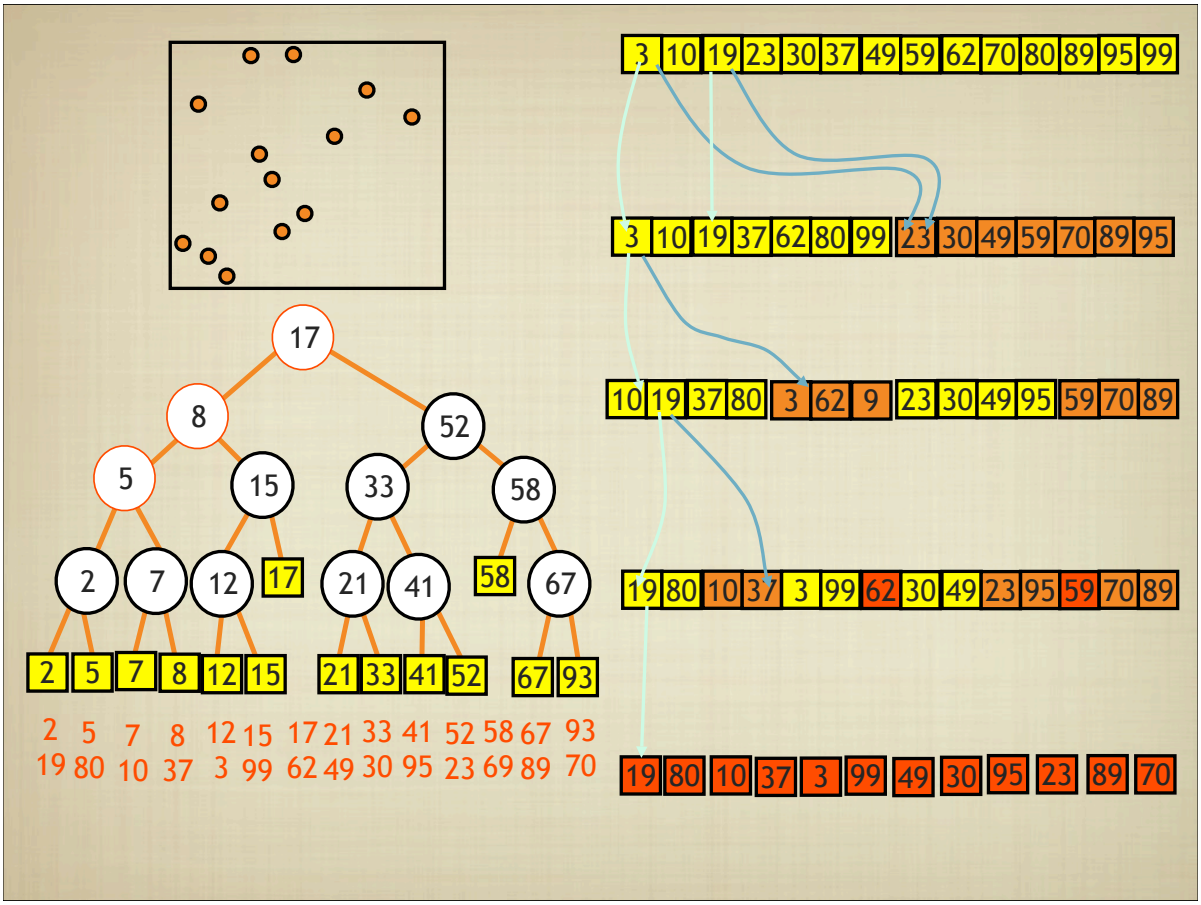


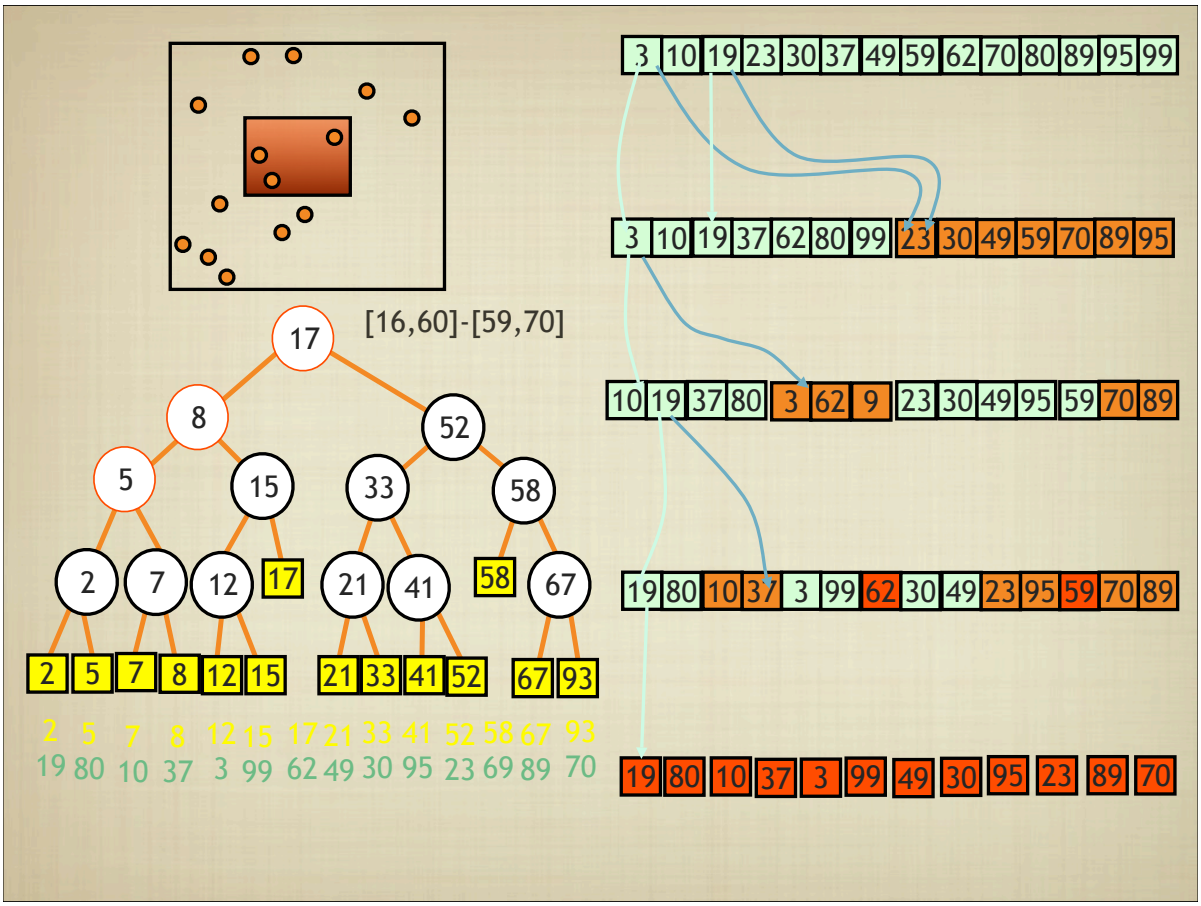
Exemplo: [20:65]

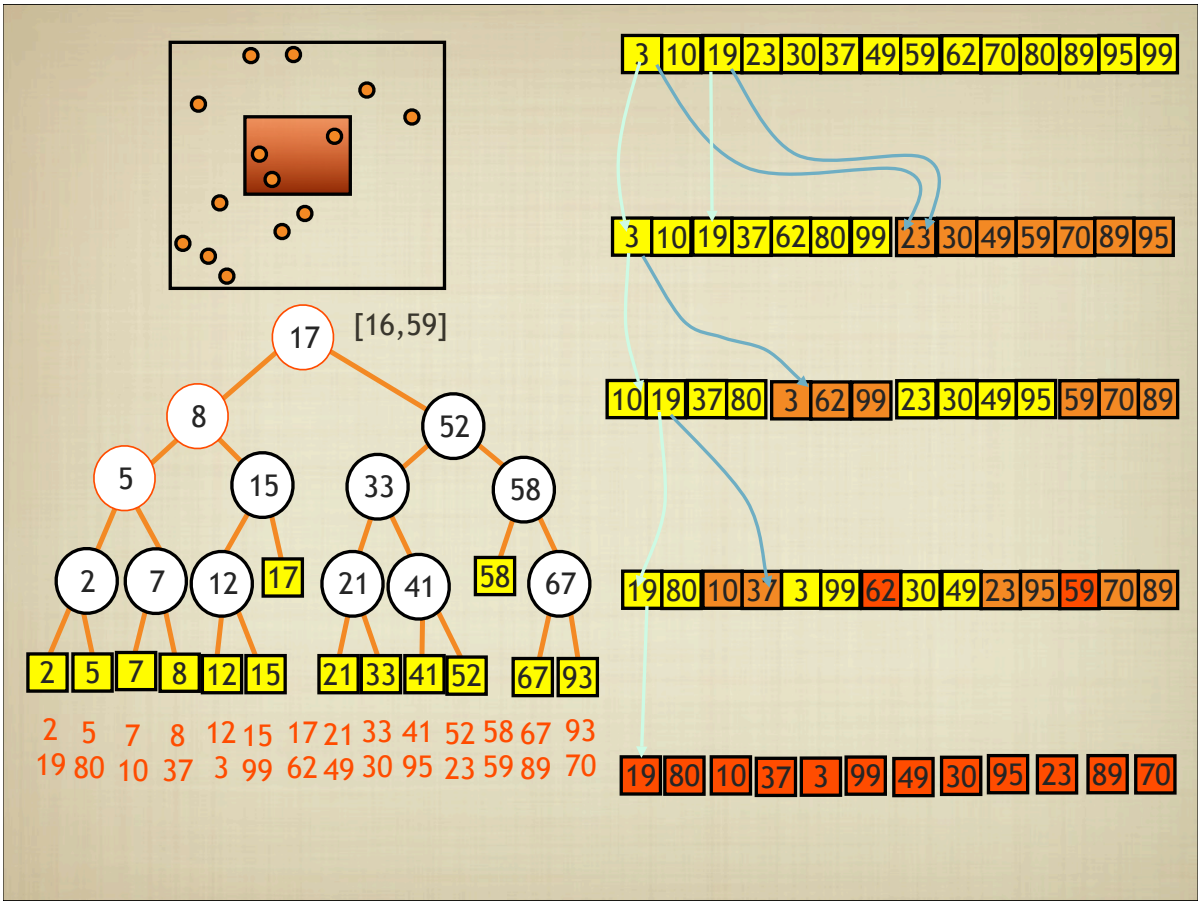
# CASCADEAMENTO FRACIONÁRIO

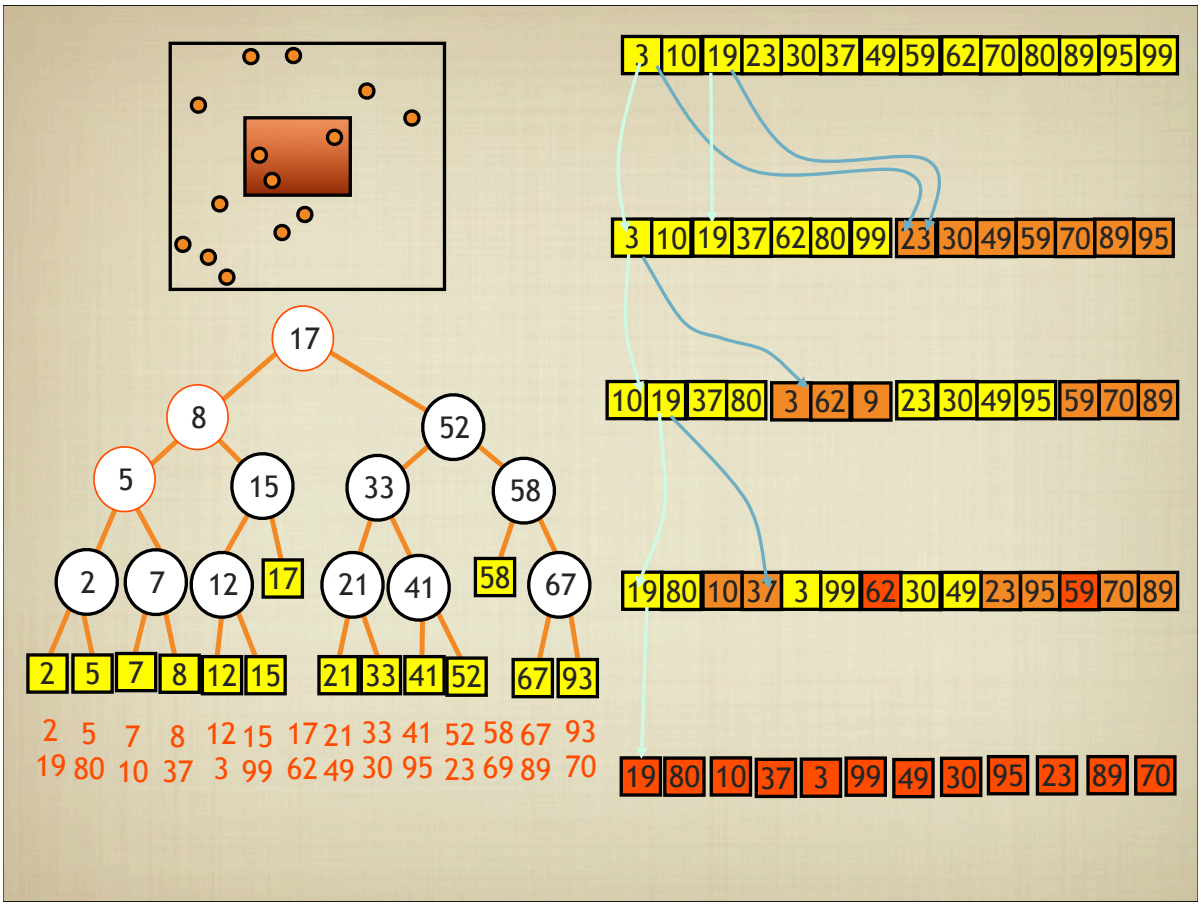



Exemplo: [20:65]




[16,60]-[59,70]
17
8
52
5
15
33
58
2
7
12
17
21
41
58
67
2 5 7 8 12 15 21 33 41 52 67 93
2 5 7 8 12 15 17 21 33 41 52 58 67 93
19 80 10 37 3 99 62 49 30 95 23 69 89 70
3 10 19 23 30 37 49 59 62 70 80 89 95 99
3 10 19 37 62 80 99 23 30 49 59 70 89 95
10 19 37 80 3 62 9 23 30 49 95 59 70 89
19 80 10 37 3 99 62 30 49 23 95 59 70 89
19 80 10 37 3 99 49 30 95 23 89 70


[16,59]
17
8
52
5
15
33
58
2
7
12
17
21
41
58
67
2 5 7 8 12 15 21 33 41 52 67 93
2 5 7 8 12 15 17 21 33 41 52 58 67 93
19 80 10 37 3 99 62 49 30 95 23 59 89 70
3 10 19 23 30 37 49 59 62 70 80 89 95 99
3 10 19 37 62 80 99 23 30 49 59 70 89 95
10 19 37 80 3 62 99 23 30 49 95 59 70 89
19 80 10 37 3 99 62 30 49 23 95 59 70 89
19 80 10 37 3 99 49 30 95 23 89 70




# CASCADEAMENTO FRACIONÁRIO