Lecture 4 Brute-Force

CS330 - Algorithm Analysis

- Brute-Force
  - Maximum/minimum Searching
  - Sorting
  - Closest Pairing
  - Convex Hull
- Exhaustive search
  - Assignment problem
- Back tracking
  - Assignment problem
  - Subset sum problem

Maximum/minimum

```
MAX(A, N)
index=0;
for (i=1; i<N; ++i) {
        if ( A[i]>A[index] ) index = i;
}
return index;
```

- N comparisons: O(N)

Searching in a collection

```
SEARCH(A, N, Val)
 for (i=0; i<N; ++i) {
         if ( A[i] == Val ) break;
}
return i;
```

- run-time:
- best - O(1)
- worst - O(N)
- average – assuming uniform distribution of the Vals.

Avg case:

key val uniformly distributed in A.

$$T(n) = \frac{1}{N}\cdot 1 + \frac{1}{N}\cdot 2 + \cdots + \frac{1}{N} N$$

$$= \frac{1}{N}(1+2+3+\cdots N)$$

$$= \frac{1}{N}\cdot\frac{(1+N)\cdot N}{2} \in \Theta(n)$$

Selection sort

1 : i=0 N−i = n

2 : i=1 N−i = n−1

n−1 : i=n−2 N−i = n − n+2 = 2

Σ

basic operation

comparison

operation in max()

SORT_SELECTION(A, N)

for (i=0; i<N-1; ++i) {

  j = MAX(A[i..N-1],N-i);//find maximum of the tail of A from index i

  SWAP(A[i], A[j]);

}

$$T(n) = 2 + 3 + \dots + n-1 + n$$

$$= \frac{(2+n)(n-1)}{2} = \frac{2n - 2 + n^2 - n}{2}$$

$$\in \Theta(n^2)$$

- run-time complexity
-   N + N-1 + N-2 + ... + 2 = (2+N)(N-1)/2 ∈ O(N^2)

0 1 2 3 4 5 6

A = [89 45 68 90 29 34 17]

0

\#1 $i \Leftarrow \max(A[0..6], 7)$

swap $(A[0], A[i=3])$.

1

\#2. $i \Leftarrow \max(A[1..6], 6)$

swap $(A[1], A[i])$

\#3: $i \Leftarrow \max(A[2..6], 5)$

⋮

n-2.

Descending order.

0 1 2 3 4 5 6

90 45 68 89 29 34 17

90 89 68 45 29 34 17

90 89 68 45 29 34 17

90 89 68 45 29 34 17

90 89 68 45 34 29 17

$\boxed{90\ 89\ 68\ 45\ 34\ 29}\ 17$

$\{ \underline{P_1^0} \ \underline{P_2^1} \ \underline{P_3^2} \ \underline{P_4^3} \}$

closest-pair

$\boxed{\begin{array}{ll} dist(P_1, P_2) & dist(P_2, P_3) \\ dist(P_1, P_3) & dist(P_2, P_4) \\ dist(P_1, P_4) & dist(P_3, P_4) \end{array}}$

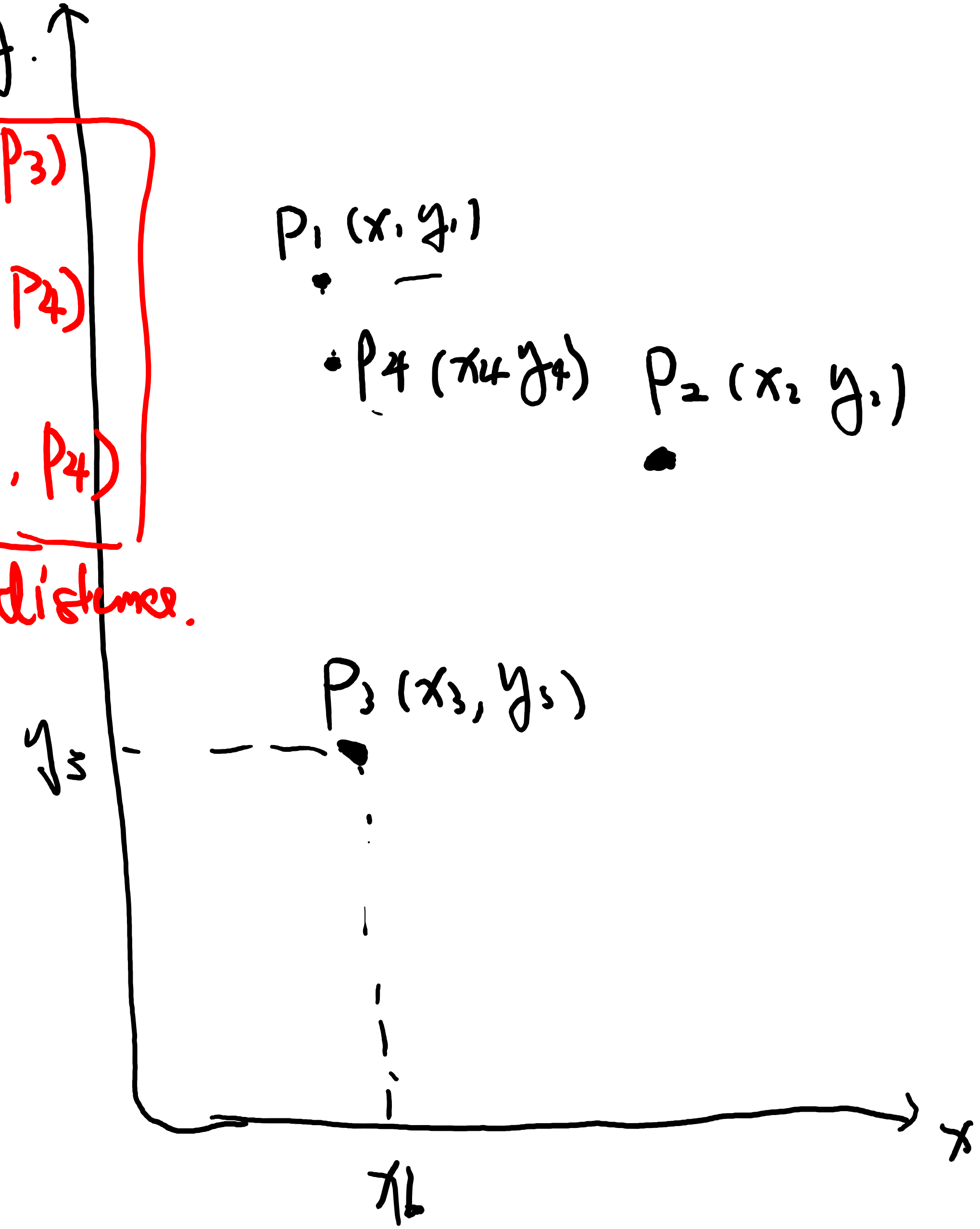
↓ minimal distance.

```
CLOSEST_PAIR(P,N)
ind1 = 0;
ind2 = 1;
min_dist = dist( P[0], P[1] )
for (i=0; i<N-1; ++i)
        for (j=i+1; j<N; ++j)
            if ( dist( P[i], P[j] ) < min_dist ) {
                ind1 = i;
                ind2 = j;
                min_dist = dist( P[i], P[j] )
            }
```

$i = 0 .. n-2$

$i+1 .. n-1$

$T(n) \in \Theta(n^2)$



time

dist( )

basic operation

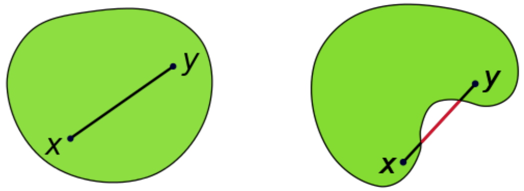
$i=0$:	$j=i+1=1 \ldots n-1$		$n-1$
$i=1$:	$j=2 \ldots$	$n-1$	$n-2$.
$i=2$:	$j=3$	$n-1$	$n-3$.
$\vdots$			
$i=n-2$	$j=n-1$	$n-1$	1

Σ.

$$T(n) = 1+2+\cdots(n-1) = \frac{(1+n-1)(n-1)}{2} \in \Theta(n^2)$$

Convex-hull(2D)

a **convex region** is a region where, for every pair of points within the region, every point on the straight line segment that joins the pair of points is also within the region



a **convex hull** of a set S of points is the smallest convex set containing S.
(smallest: the convex hull of S must be a subset of any convex set containing S.)

CONVEX_HULL_WRONG(P, N)
for all subsets S of P
if S is a convex hull
return S

CONVEX_HULL1(P, N)
for all subsets S of P
for all permutations PS of S
if PS is a convex hull
return PS

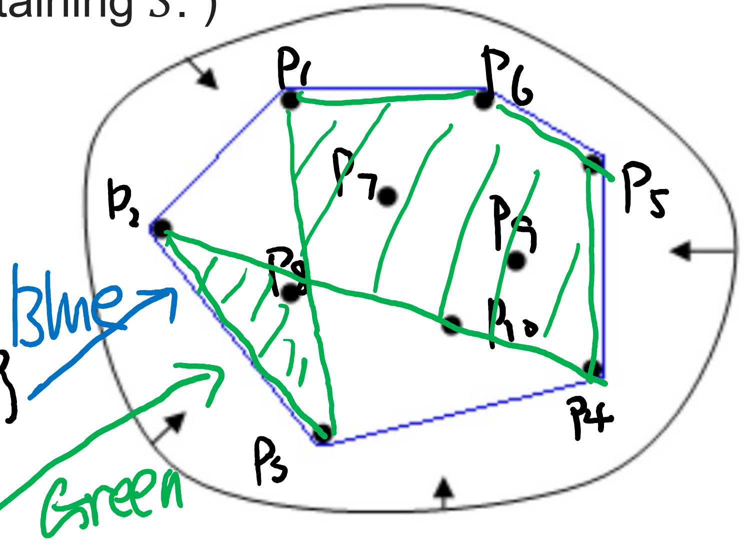
$P = \{P_1, P_2, \ldots, P_n\}$

subset of P is

$\{P_1, P_2, \ldots P_6\}$

convex hull ← Permutation $PS1 = \{P_1 P_2 P_3 P_4 P_5 P_6\}$

not a convex hull ← Permutation $PS2 = \{P_1 P_3 P_2 P_4 P_5 P_6\}$



Convex-hull (New info)

- edge P[i],P[j] belongs to convex hull iff all other points P[k] lie on the same side of the line P[i],P[j]

```
CONVEX_HULL3( P, N )
S=0; //set of points forming convex hull
for (i=0; i<N-1; ++i) {
        for (j=i+1; j<N; ++j) {
                same_side = true;
                for (k=0; k<N && same_side; ++k) {
                        same_side = same_side && ( P[k] is on same side with other points)
                }
                if ( same_side ) add P[i] and P[j] to S
        }
}
return S
```

$O(N^3)$

Exhaustive search

Brute force is a straightforward approach to solving a problem, usually directly based on the problem statement and definitions of the concepts involved.

- Exhaustive search is an important special case of Brute force.
- To combinatorial problems
  - Convex Hull 1

Job Assignment Problem

- n people, n jobs, one person per job
- Cost[i,j] – the cost of person ***i*** is assigned to job ***j***
- To find an assignment with minimum total cost

	Job 1	**Job 2**	**Job 3**	**Job 4**
Person 1	9	2	7	8
Person 2	6	4	3	7
Person 3	5	8	1	8
Person 4	7	6	9	4

Tuples

- 4 person: p1, p2, p3, p4
- 4 jobs: J1, J2, J3, J4
- <1 2 3 4> : person ***p1*** is assigned job ***J1***, …
  - Index: person, value: assignment
  - <2, 4, 1, 3>: person ***p1*** is assigned job ***J2***, …
- Total cost of <1 2 3 4>: 9+4+1+4=18

COST

	Job 1	**Job 2**	**Job 3**	**Job 4**
Person 1	9	2	7	8
Person 2	6	4	3	7
Person 3	5	8	1	8
Person 4	7	6	9	4

How many tuples

- 1 2 3 4
- 1 3 2 4
- …
- Permutation

- Cost for 4! tuples should be calculated

- n people n jobs: n!

```cpp
for (int i1 = 0; i1 < 3; ++i1) {
  for (int i2 = 0; i2 < 3; ++i2) {
    for (int i3 = 0; i3 < 3; ++i3) {
      if (i1!=i2 && i1!=i3 && i2!=i3) {
        std::cout << "{" << array[i1] << ","
          << array[i2] << "," << array[i3] << "}\n";
      }
    }
  }
}
```

COST

	Job 1	Job 2	Job 3	Job 4
Person 1	9	2	7	8
Person 2	6	4	3	7
Person 3	5	8	1	8
Person 4	7	6	9	4

Solution space

person1

person2

person3

person4

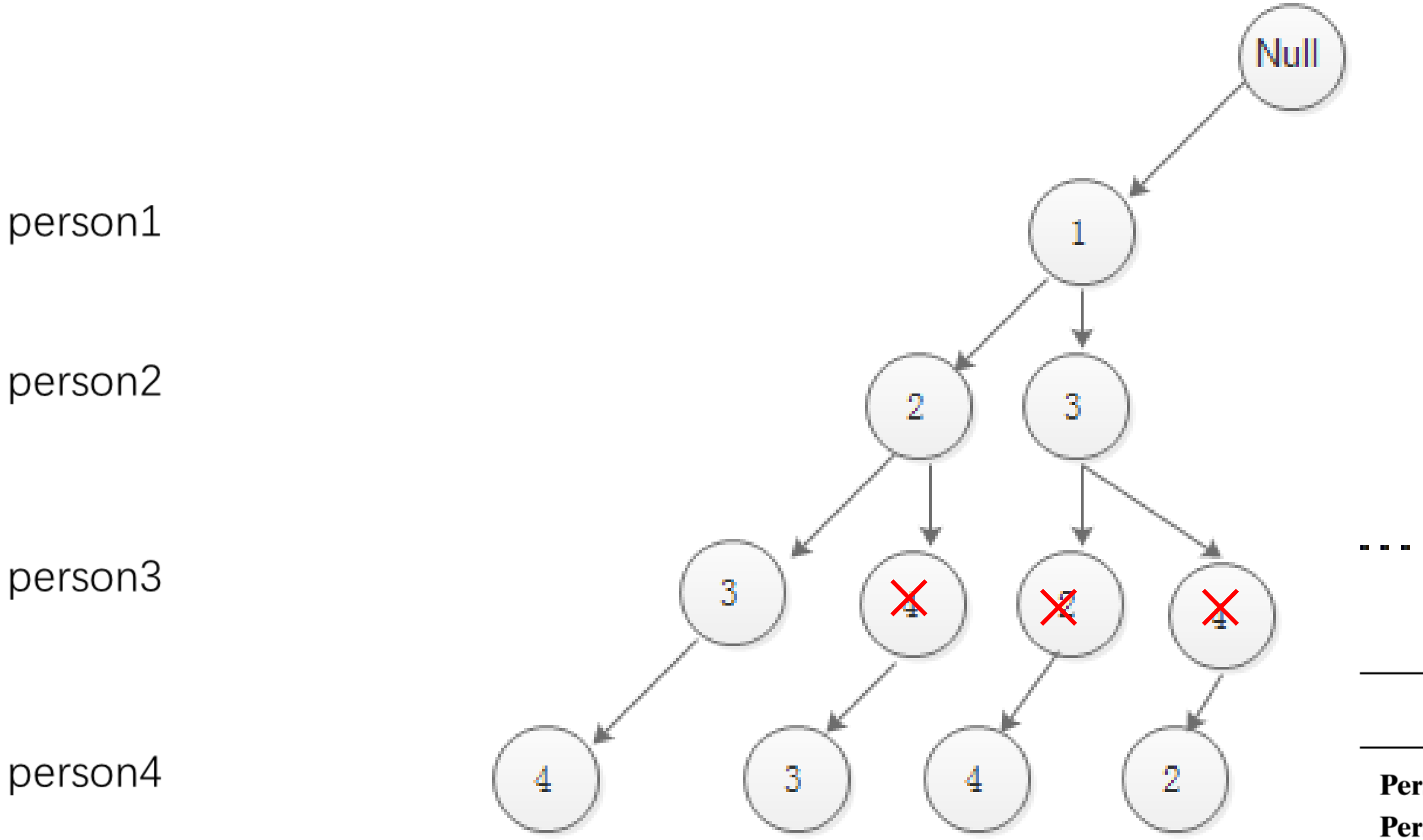
Back-tracking: list all solutions

```
Algorithm work
Global: n, cost[n][n]
Input: person i, counts, solution
Result: return all feasible solutions and costs
{
    If size of solution is n
        output solution & total cost and return

    for (j=1..n)
        if job j hasn't been taken
            add job j to solution
            work(i+1, counts+=cost[i][j], solution)
            [delete job j from solution]    }
```

Work(1, 0, null Solution)

Backtracking with bounding value

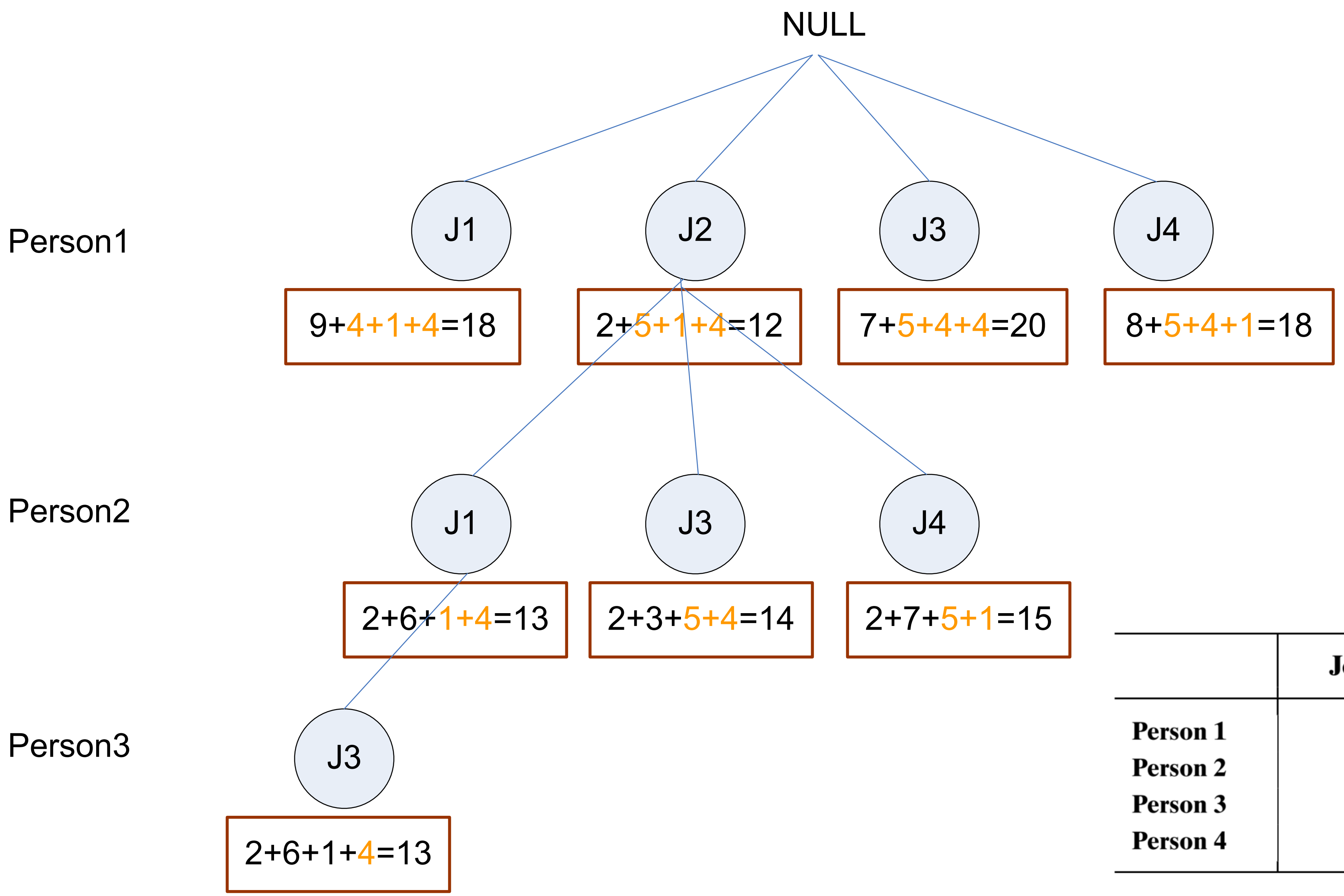


Bounding = 18

COST

	Job 1	Job 2	Job 3	Job 4
Person 1	9	2	7	8
Person 2	6	4	3	7
Person 3	5	8	1	8
Person 4	7	6	9	4

Branch and Bound with Best First



COST

	Job 1	**Job 2**	**Job 3**	**Job 4**
Person 1	9	2	7	8
Person 2	6	4	3	7
Person 3	5	8	1	8
Person 4	7	6	9	4

Subset-sum problem

- find a subset of {3,4,5,6} whose elements add up to 15