# Greedy Algorithms

Meng-Tsung Tsai

02/05/2018

# A common technique

Here is a common technique to design greedy algorithms.

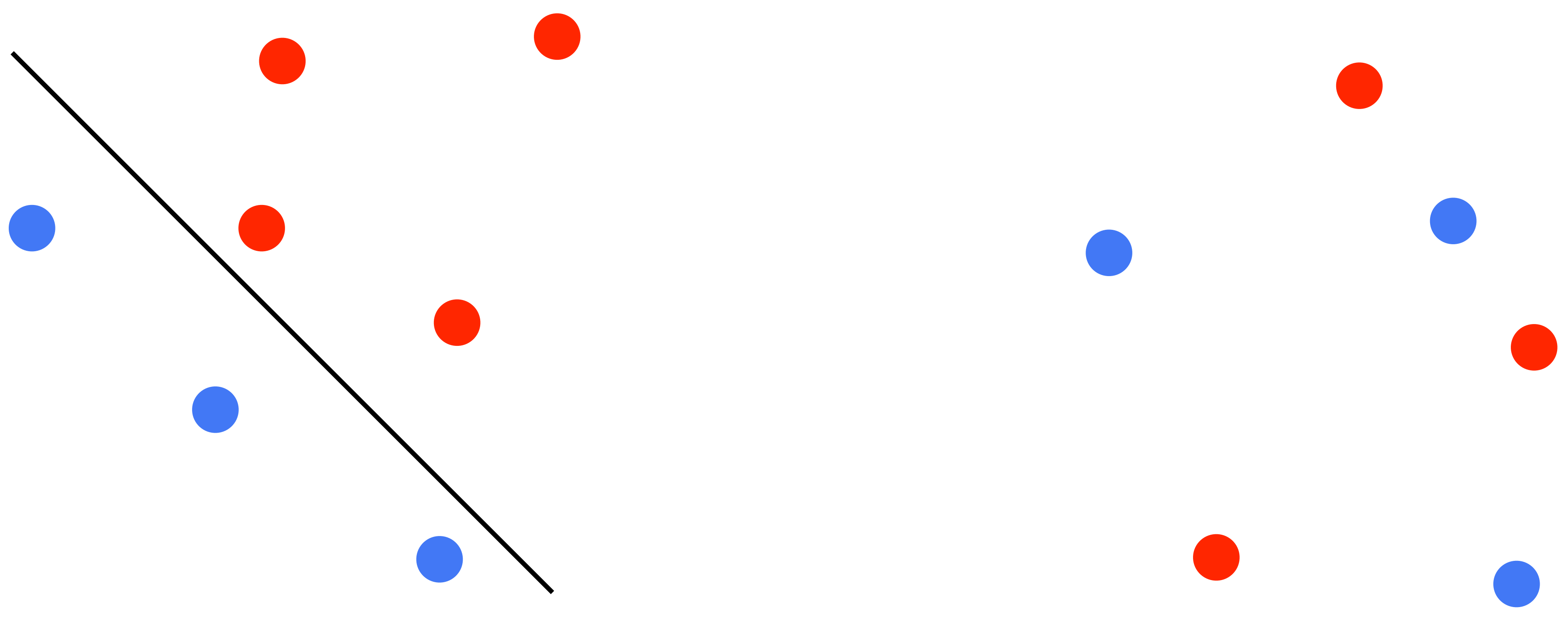
(1) Imagine what the optimum solution is.
(2) Give an initial guess.
(3) If the guess $\neq$ the optimum solution, find a way to **morph** the guess to the solution.

**G. W. Busch**

**A. Schwarzenegger**

# Seperate colored points by a line

Given n points on a 2D plane. The color of each point is blue or red. Decide whether there exists a line that seperates the points into two halves so that each half contains points of same color.

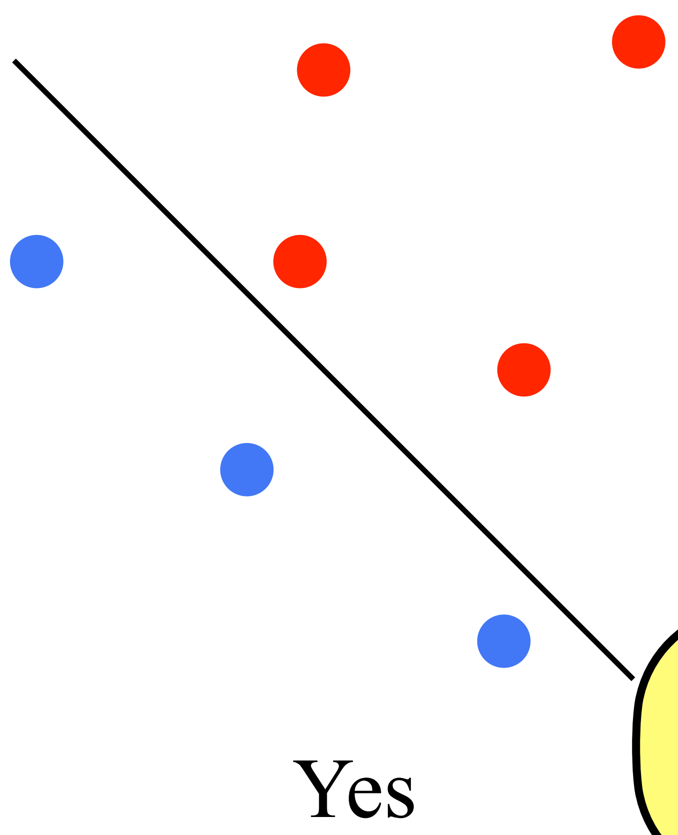


Yes

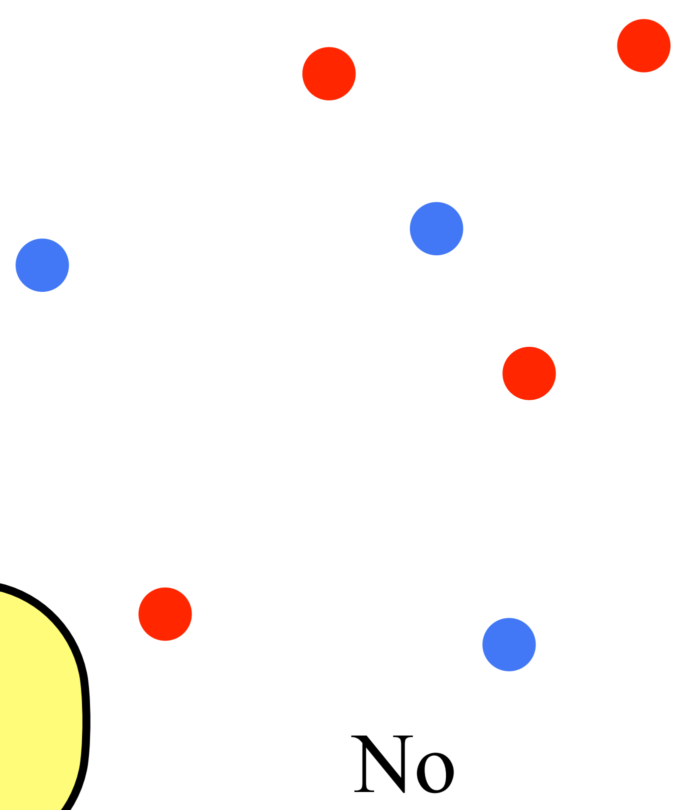
No

# Seperate colored points by a line

Given n points on a 2D plane. The color of each point is blue or red. Decide whether there exists a line that seperates the points into two halves so that each half contains points of same color.

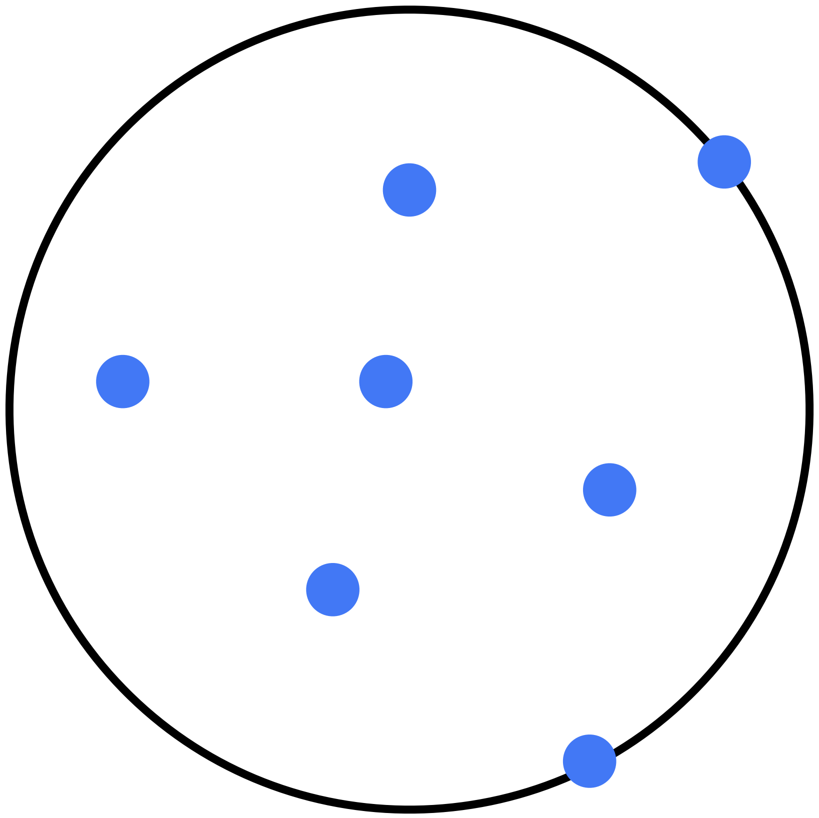



O($n^3$) time.
Optimal?



# The minimum enclosing ball problem

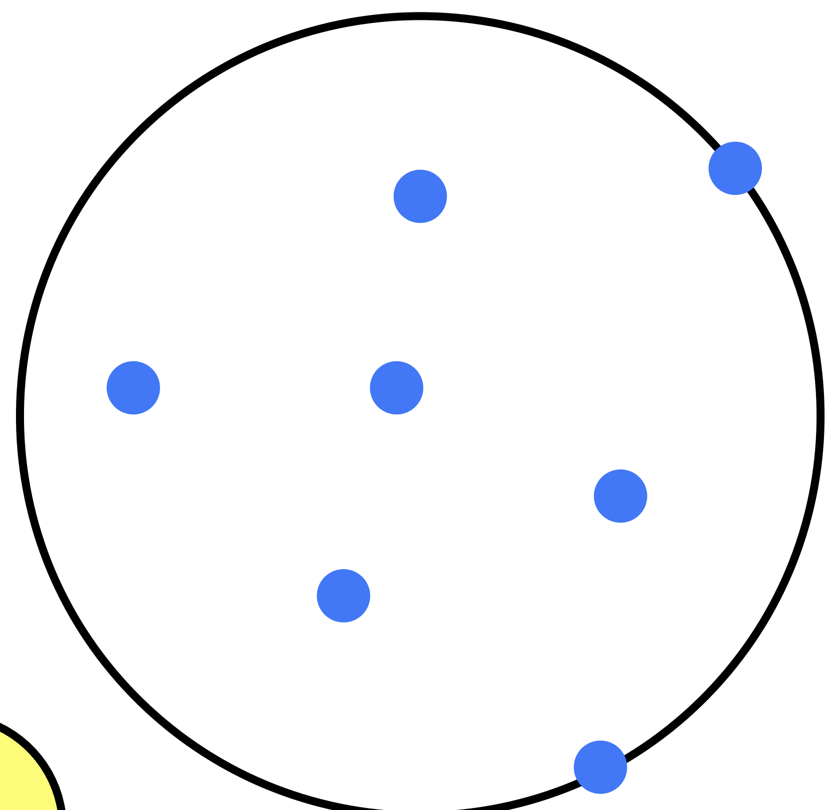
Given n points on a 2D plane, find the smallest circle so that each of the n point is either on the boundary of the circle or in its interior.



An enclosing ball

# The minimum enclosing ball problem

Given n points on a 2D plane, find the smallest circle so that each of the n point is either on the boundary of the circle or in its interior.

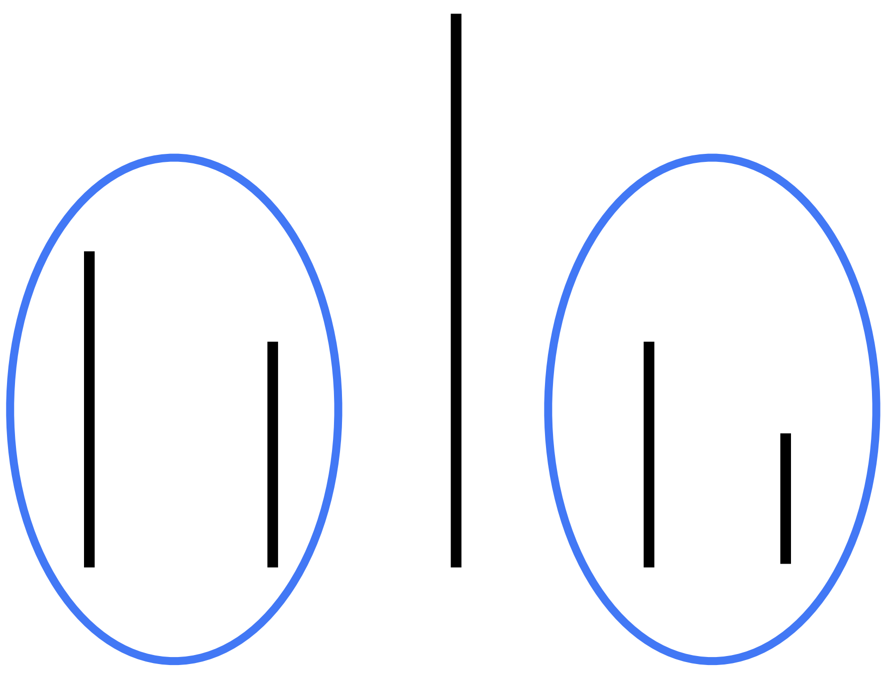


An enclosing ball

$O(n^4)$ time.
Optimal?

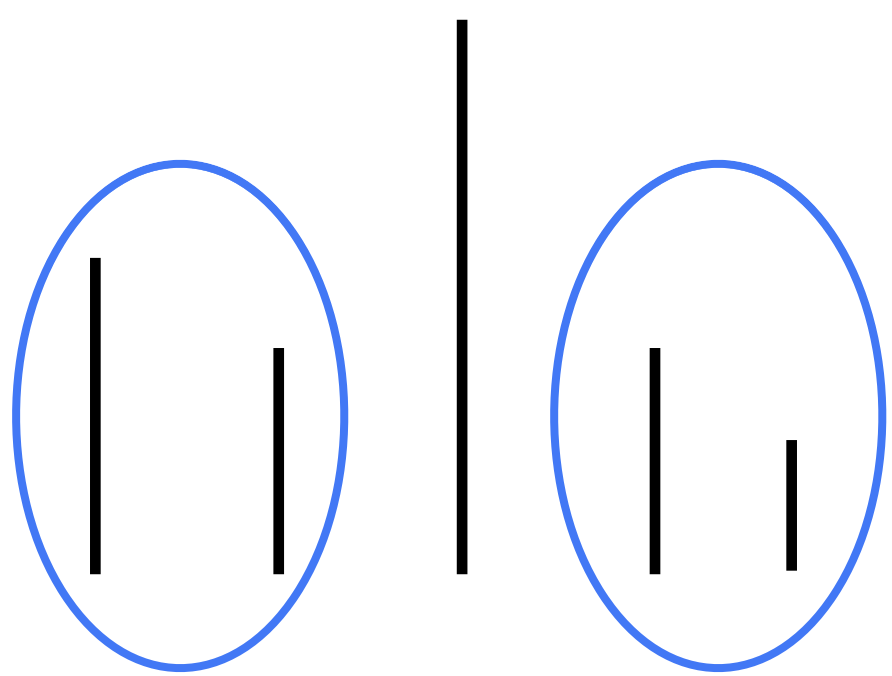
# Pair Chopsticks

Given n chopsticks that have length $\ell_1, \ell_2, \ldots, \ell_n \geq 0$. If two chopsticks have length difference smaller than d, then we can pair the two chopsticks. Maximize the number of paired chopsticks, noting that each chopstick can join at most 1 pair.



# Pair Chopsticks

Given n chopsticks that have length $\ell_1, \ell_2, \ldots, \ell_n \geq 0$. If two chopsticks have length difference smaller than d, then we can pair the two chopsticks. Maximize the number of paired chopsticks, noting that each chopstick can join at most 1 pair.



O(n log n) time.

# Another common technique

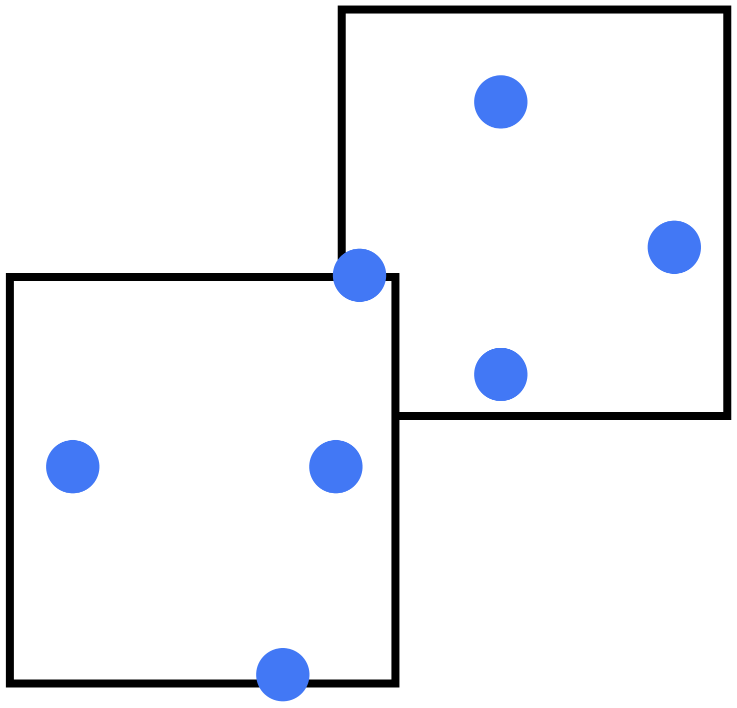
Here is another common technique to design greedy algorithms.

(1) Let some parameter be a fixed value and see whether you can solve the problem.

(2) Observe the relationship (e.g. monotonicity) between the solutions for two different fixed values.

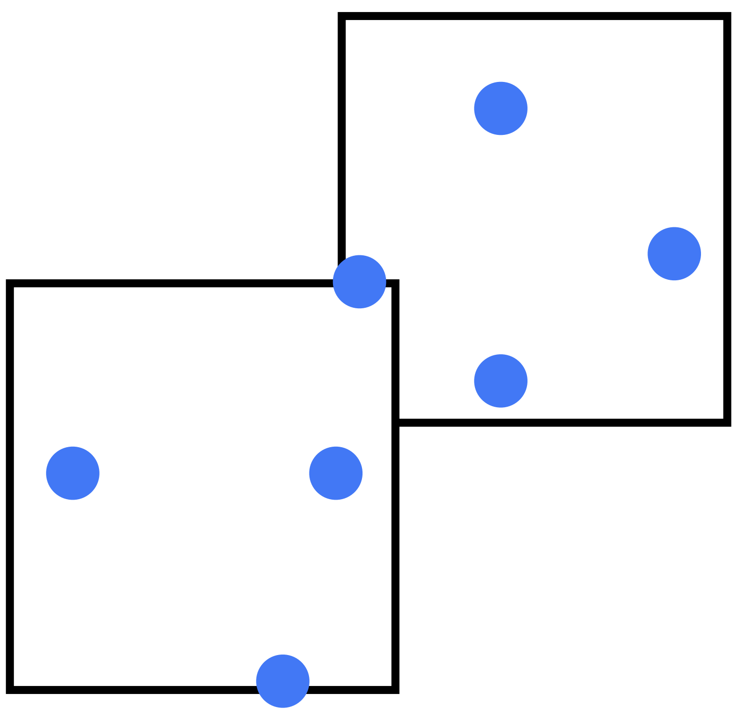
# Cover points by two sqaures of length L

Given n points on a 2D plane, cover all the points by two squares of length L so that L is minimized. These two squares may overlap.



# Cover points by two sqaures of length L

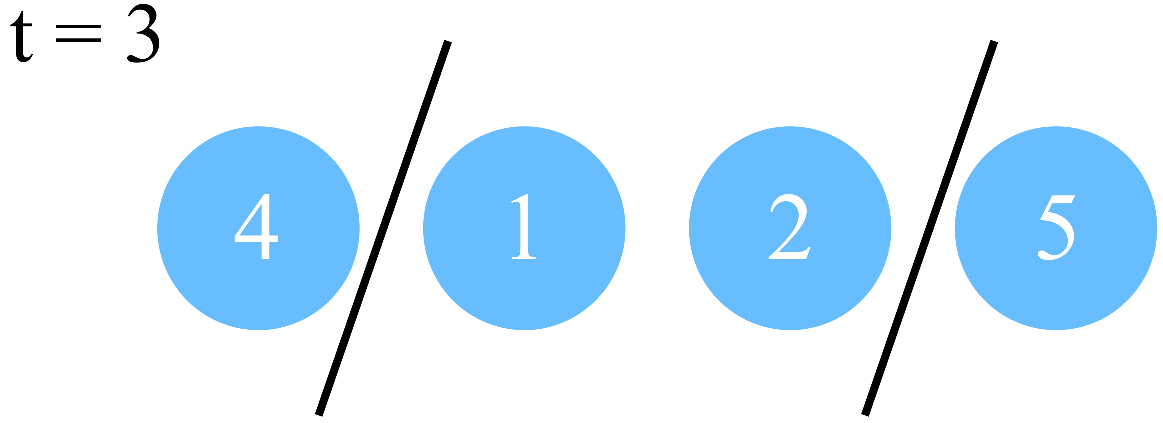
Given n points on a 2D plane, cover all the points by two squares of length L so that L is minimized. These two squares may overlap.



$O(n^2 \log n)$ time.
What if L must be an integer?

# Partition a sequence

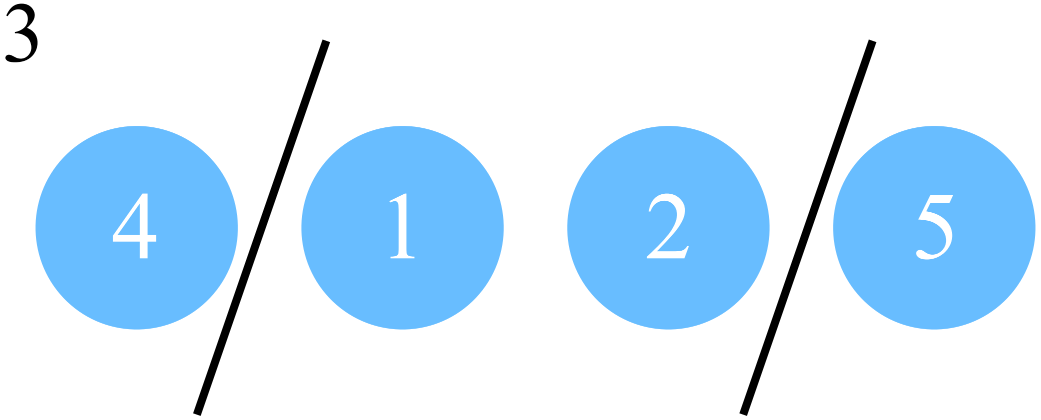
Given a sequence of n positive integers $a_1, a_2, \ldots, a_n$. Define weight of a subsequence to be the sum of elements in it. Partition these n integers into t consecutive subsequences so that the maximum weight of the t subsequences is minimized.



# Partition a sequence

Given a sequence of n positive integers $a_1, a_2, \ldots, a_n$. Define weight of a subsequence to be the sum of elements in it. Partition these n integers into t consecutive subsequences so that the maximum weight of the t subsequences is minimized.

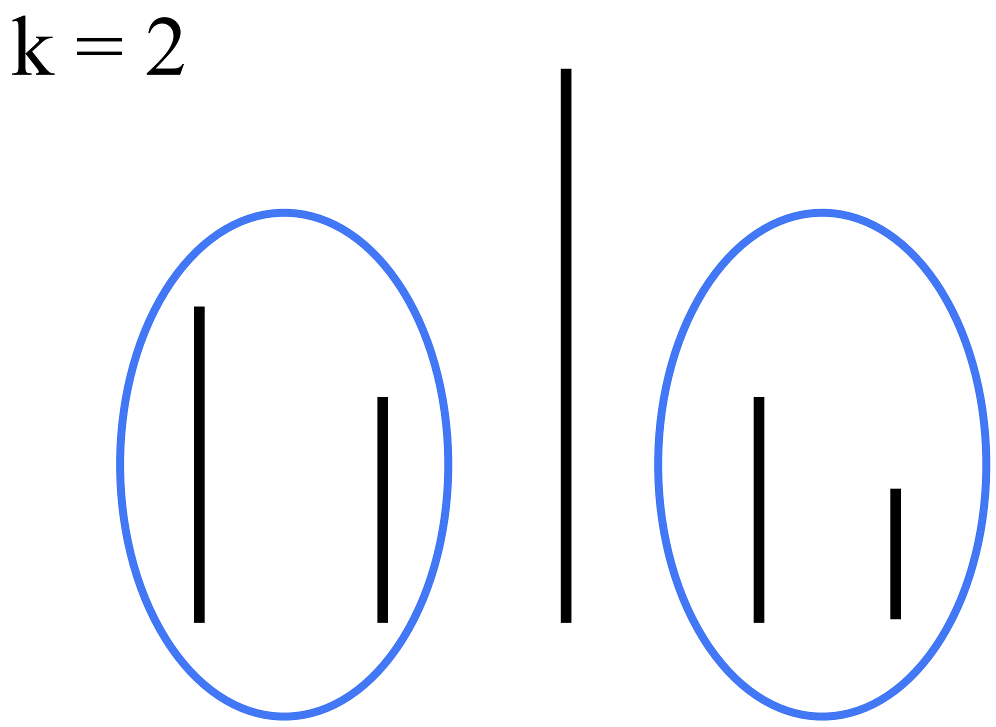
$t = 3$




$O(n \log(\max_i a_i))$ time.

# Pair Chopsticks (A variation)

Given n chopsticks that have **integral** length $\ell_1, \ell_2, \ldots, \ell_n \geq 0$. Find k pairs of chopsticks from the n given ones so that the maximum length difference in a pair is minimized.

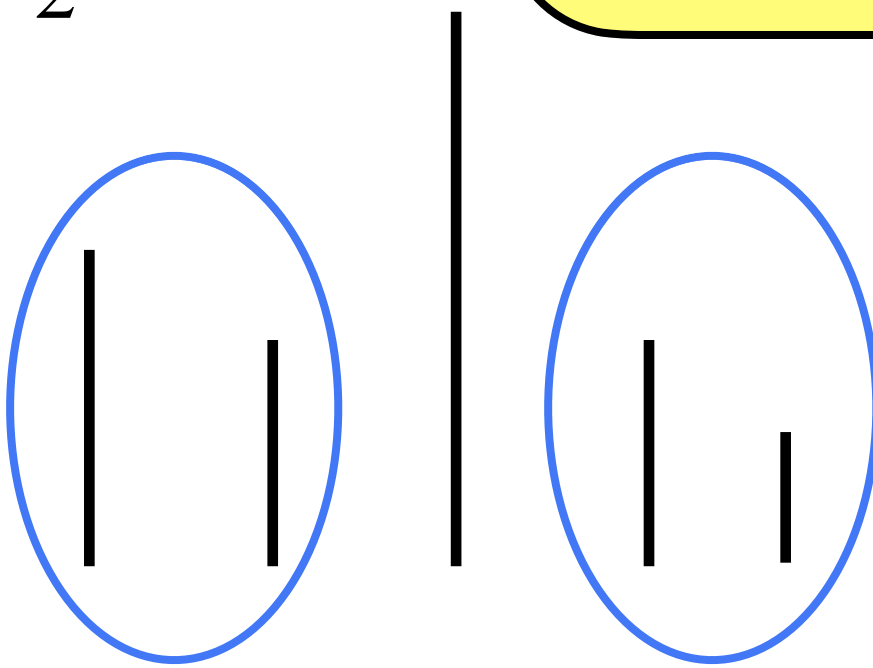


# Pair Chopsticks (A variation)

Given n chopsticks that have **integral** length $\ell_1, \ell_2, \ldots, \ell_n \geq 0$. Find k pairs of chopsticks from the n given ones so that the maximum length difference in a pair is minimized.

$O(n\,(\log n + \log(\max_i \ell_i)))$ time.

k = 2



# Bridge

n people would like to cross a bridge. A group of at most two people may cross the bridge at any time with a flashlight. There is only one flashlight. People $p_i$ needs $t_i$ seconds to cross the bridge. If $p_i$ and $p_j$ cross the bridge together, then they needs $\max(p_i, p_j)$ seconds. Find an arrangment so that it takes the least time for all to reach the other side of the bridge.

# Bridge

n people would like to cross a bridge. A group of at most two people may cross the bridge at any time with a flashlight. There is only one flashlight. People $p_i$ needs $t_i$ seconds to cross the bridge. If $p_i$ and $p_j$ cross the bridge together, then they needs $\max(p_i, p_j)$ seconds. Find an arrangment so that it takes the least time for all to reach the other side of the bridge.

O(n log n) time.

# Matroid

# What is a matriod?

It is a structure $M = (S, F)$, where:

(1) $S$ is a set of $n$ elements
(2) $F$ is a non-empty collection of subsets of $S$

<u>Example</u>.

Given $S = \{1, 2, .., n\}$

F could be {
$S_1 = \{2\}$,
$S_2 = \{1, n\}$,
$S_3 = \varnothing$,
…}

# What is a matriod?

It is a structure $M = (S, F)$, where:

(1) S is a set of n elements
(2) F is a non-empty collection of subsets of S

(3) F is **hereditary**. That is, if $A \subset B$ and $B \in F$, then $A \in F$.

If $F = \{$
$S_1 = \{1, n\}, \ldots\}$

Then, F contains also
$\varnothing$, $\{1\}$ **and** $\{n\}$.

# What is a matriod?

It is a structure $M = (S, F)$, where:

(1) $S$ is a set of $n$ elements
(2) $F$ is a non-empty collection of subsets of $S$

(3) $F$ is **hereditary**. That is, if $A \subset B$ and $B \in F$, then $A \in F$.
(4) $M$ satisfies the **exchange property**. That is, for any $A, B \in F$, if $|A| < |B|$, then exists $x \in B \setminus A$ so that $A \cup \{x\} \in F$.

If $F = \{$
$S_1 = \{2\}$,
$S_2 = \{1, n\}, \ldots\}$

Then, $F$ contains also $\{1, 2\}$ **or** $\{2, n\}$.

# What is a matroid?

It is a structure $M = (S, F)$, where:

(1) $S$ is a set of $n$ elements
(2) $F$ is a non-empty collection of subsets of $S$

(3) $F$ is **hereditary**. That is, if $A \subset B$ and $B \in F$, then $A \in F$.
(4) $M$ satisfies the **exchange property**. That is, for any $A, B \in F$, if $|A| < |B|$, then exists $x \in B \setminus A$ so that $A \cup \{x\} \in F$.

Such a structure $M$ is called a **matroid**.
We say any subset in $F$ is an **independent** subset.

# The Weighted Matriod Problem

# A non-negative weight function

A function w that assigns a **non-negative** weight w(x) to each element x in S.

For any subset R of S, we define $w(R) = \sum_{e \in R} w(e)$.

# A non-negative weight function

A function w that assigns a **non-negative** weight w(x) to each element x in S.

For any subset R of S, we define $w(R) = \sum_{e \in R} w(e)$.

We say M = (S, F) is a **weighted matroid** if its S is associated with a weight function $w: S \rightarrow R_+ \cup \{0\}$.

# The weighted matroid problem

Input: a matroid $M = (S, F)$ and a weight function $w: S \rightarrow R_+ \cup \{0\}$.

Output: an independent subset $R$ in $F$ so that $w(R)$ is maximized.

# The weighted matroid problem

Input: a matroid $M = (S, F)$ and a weight function $w: S \rightarrow R_+ \cup \{0\}$.

Output: an independent subset $R$ in $F$ so that $w(R)$ is maximized.

Many problems can be encoded as a weighted matroid problem. We can solve all such problems in a unified way.

# The greedy algorithm that solves the weighted matroid problem optimally

```
Greedy(M = (S, F), w){
    Sort the elements in S into the order of non-increasing
    weight w. Let the sorted order be s1, s2, …, sn.

    A = ∅; // let the answer be an empty set initially
    for(i=1; i≤n; ++i){
        if(A ∪ {si} in F){
            A = A ∪ {si};
        }
    }
    return A;
}
```

# The greedy algorithm that solves the weighted matroid problem optimally

```
Greedy(M = (S, F), w){
    Sort the elements in S into the order of non-increasing
    weight w. Let the sorted order be s1, s2, …, sn.

    A = ∅; // let the answer be an empty set initially
    for(i=1; i≤n; ++i){
        if(A ∪ {si} in F){
            A = A ∪ {si};
        }
    }
    return A;
}
```

That is it. Now we have a unified solution for any weighted matroid problem.

# Applications

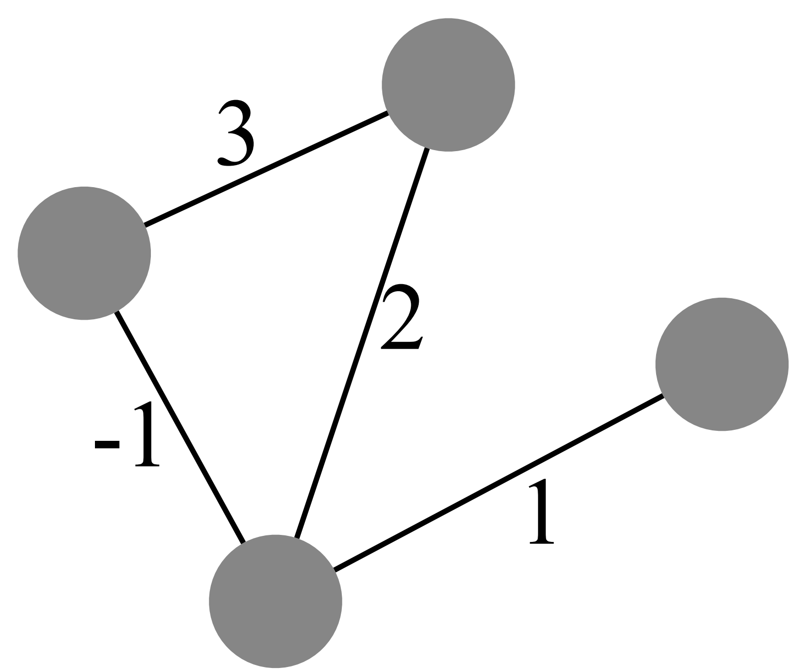
# The Minimum Spanning Tree Problem

Input: a connected undirected graph $G = (V, E)$ and a weight function $w: E \rightarrow R$.

Ouput: a spanning tree $T$ of $G$, i.e. a tree containing all nodes, so that $w(T) = \sum_{e \in T} w(e)$ is minimized.

<u>Example</u>.

graph G

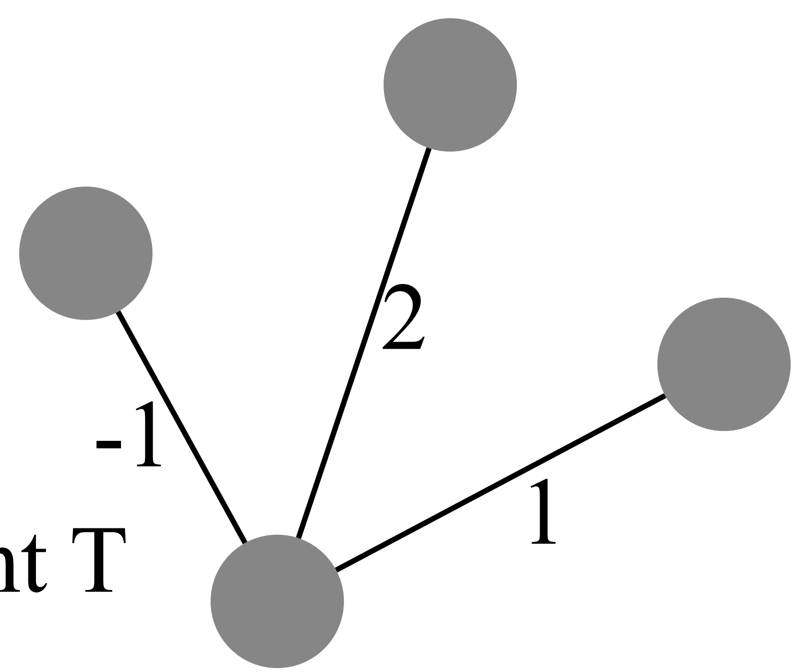


# The Minimum Spanning Tree Problem

Input: a connected undirected graph G = (V, E) and a weight function w: E → R.

Ouput: a spanning tree T of G, i.e. a tree containing all nodes, so that $w(T) = \sum_{e \in T} w(e)$ is minimized.

<u>Example</u>.




an optimum-weight T

# MST is a weighted matroid problem (1/3)

Plan: encoding the MST as a weighted matriod problem.
Thus, we can solve it by the unified solution of WMP.

We define a matriod as follows.

Let $M_G = (E, F)$.

(1) E is the set of all edges in G.
(2) F is the set of all spanning forests of G, i.e. a subset of E.

# MST is a weighted matroid problem (1/3)

Plan: encoding the MST as a weighted matriod problem.
Thus, we can solve it by the unified solution of WMP.

We define a matriod as follows.

Let $M_G = (E, F)$.

(1) E is the set of all edges in G.
(2) F is the set of all spanning forests of G, i.e. a subset of E.
(3) Is F hereditary?

# MST is a weighted matroid problem (1/3)

Plan: encoding the MST as a weighted matriod problem.
Thus, we can solve it by the unified solution of WMP.

We define a matriod as follows.

Let $M_G = (E, F)$.

(1) E is the set of all edges in G.
(2) F is the set of all spanning forests of G, i.e. a subset of E.
(3) Is F hereditary?

Given the edge set B of a spanning forest of G (i.e. a subgraph containing no cycles), then of course any subset A of B is a spanning forest.

# MST is a weighted matroid problem (2/3)

The last question is whether $M_G$ satisfies the exchange property. If yes, then $M_G$ is a matroid.

# MST is a weighted matroid problem (2/3)

The last question is whether $M_G$ satisfies the exchange property. If yes, then $M_G$ is a matroid.

(4) Does $M_G$ satisfy the exchange property?

# MST is a weighted matroid problem (2/3)

The last question is whether $M_G$ satisfies the exchange property. If yes, then $M_G$ is a matroid.

(4) Does $M_G$ satisfy the exchange property?

Proof. Let A, B be the edge set of two spanning forests and $|A| < |B|$. Thus, # connected components in A is more than # connected components in B. Then, B must contain a tree T whose nodes are in two different trees in A. Since T is connected, then it must contain an edge $\{u, v\}$ that connects two different components in A. In other words $A \cup \{u, v\}$ is a spanning forest, i.e. in F.

# MST is a weighted matroid problem (2/3)

The last question is whether $M_G$ satisfies the exchange property. If yes, then $M_G$ is a matroid.

(4) Does $M_G$ satisfy the exchange property?

Proof. Let A, B be the edge set of two spanning forests and $|A| < |B|$. Thus, # connected components in A is more than # connected components in B. Then, B must contain a tree T whose nodes are in two different trees in A. Since T is connected, then it must contain an edge $\{u, v\}$ that connects two different components in A. In other words $A \cup \{u, v\}$ is a spanning forest, i.e. in F.

$M_G$ is a matroid.

# MST is a weighted matroid problem (3/3)

In MST, we want to find a spanning tree of the **minimum weight**. In a weighted matroid problem, it returns an independent subset of the **maximum weight**. Since they are mismatched, we need a fix here.

# MST is a weighted matroid problem (3/3)

In MST, we want to find a spanning tree of the **minimum weight**. In a weighted matroid problem, it returns an independent subset of the **maximum weight**. Since they are mismatched, we need a fix here.

Solution. We can let $w_0$ be the $\max_{e \in E} w(e)$, and define a new weight function $z(e) = w_0 - w(e)$. In this way, the weighted matriod problem with input $M_G = (E, F)$ and $z$ is equivalent to the MST problem.

# A task scheduling problem

Input: a set S of unit-time tasks $\{s_1, s_2, \ldots, s_n\}$. Each task $s_i$ has a deadline $d_i$ and a non-negative penalty $w_i$.

Output: a permutation of S so that the first task begins at time 0 and finishes at time 1, the second task begins at time 1 and finishes at time 2, etc. and the total **penalty** incurred by the tasks not finished by their dealines is **minimized**.

Example.

| $s_i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| $d_i$ | 4 | 2 | 4 | 3 | 1 | 4 | 6 |
| $w_i$ | 70 | 60 | 50 | 40 | 30 | 20 | 10 |

# A task scheduling problem

Input: a set S of unit-time tasks $\{s_1, s_2, \ldots, s_n\}$. Each task $s_i$ has a deadline $d_i$ and a non-negative penalty $w_i$.

Output: a permutation of S so that the first task begins at time 0 and finishes at time 1, the second task begins at time 1 and finishes at time 2, etc. and the total **penalty** incurred by the tasks not finished by their dealines is **minimized**.

penalty = 50

Example.

| $s_i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| $d_i$ | 4 | 2 | 4 | 3 | 1 | 4 | 6 |
| $w_i$ | 70 | 60 | 50 | 40 | 30 | 20 | 10 |

# TSP is a weighted matroid problem (1/3)

Plan: encoding the TSP as a weight matriod problem. Thus, we can solve it by the unified solution of WMP.

We define a matriod as follows.

Let $M_S = (S, F)$.

(1) S is the set of all tasks.
(2) F is a collection of subsets R of S so that all tasks in each R can be finished by their deadlines.

# TSP is a weighted matroid problem (1/3)

Plan: encoding the TSP as a weight matriod problem. Thus, we can solve it by the unified solution of WMP.

We define a matriod as follows.

Let $M_S = (S, F)$.

(1) S is the set of all tasks.
(2) F is a collection of subsets R of S so that all tasks in each R can be finished by their deadlines.
(3) Is F hereditary?

# TSP is a weighted matroid problem (1/3)

Plan: encoding the TSP as a weight matriod problem. Thus, we can solve it by the unified solution of WMP.

We define a matriod as follows.

Let $M_S = (S, F)$.

(1) S is the set of all tasks.
(2) F is a collection of subsets R of S so that all tasks in each R can be finished by their deadlines.
(3) Is F hereditary?

If you can solve a set of tasks all by their deadlines, then of course you can solve any subset of the tasks all by their deadlines.

# TSP is a weighted matroid problem (2/3)

The last question is whether $M_S$ satisfies the exchange property. If yes, then $M_S$ is a matroid.

# TSP is a weighted matroid problem (2/3)

The last question is whether $M_S$ satisfies the exchange property. If yes, then $M_S$ is a matroid.

(4) Does $M_S$ satisfy the exchange property?

# TSP is a weighted matroid problem (2/3)

The last question is whether $M_S$ satisfies the exchange property. If yes, then $M_S$ is a matroid.

(4) Does $M_S$ satisfy the exchange property?

Proof. Yes, see the proof on Page 445 in I2A.

# TSP is a weighted matroid problem (2/3)

The last question is whether $M_S$ satisfies the exchange property. If yes, then $M_S$ is a matroid.

(4) Does $M_S$ satisfy the exchange property?

Proof. Yes, see the proof on Page 445 in I2A.

$M_S$ is a matroid.

# TSP is a weighted matroid problem (3/3)

Since $M_S$ is a matriod, solving the corresponding weighted matriod problem $M_S = (S, F)$ and $w(s_i) = d_i$ gives a subset of the n tasks, all of which can be solved by their deadlines and the total penalty is maximized.

# TSP is a weighted matroid problem (3/3)

Since $M_S$ is a matriod, solving the corresponding weighted matriod problem $M_S = (S, F)$ and $w(s_i) = d_i$ gives a subset of the n tasks, all of which can be solved by their deadlines and the total penalty is maximized.

In other words, the penalty incurred by those tasks not finished by their deadlines is minimized, as required.

# The Correctness of the Greedy Algorithm for the Weighted Matroid Problem

# Recall the greedy algorithm

```
Greedy(M = (S, F), w){
    Sort the elements in S into the order of non-increasing
    weight w. Let the sorted order be s1, s2, …, sn.

    A = ∅; // let the answer be an empty set initially
    for(i=1; i≤n; ++i){
        if(A ∪ {si} in F){
            A = A ∪ {si};
        }
    }
    return A;
}
```

# Recall the greedy algorithm

```
Greedy(M = (S, F), w){
    Sort the elements in S into the order of non-increasing
    weight w. Let the sorted order be s1, s2, …, sn.

    A = ∅; // let the answer be an empty set initially
    for(i=1; i≤n; ++i){
        if(A ∪ {si} in F){
            A = A ∪ {si};
        }
    }
    return A;
}
```

The returned subset $A = \{a_1, a_2, \ldots, a_x\}$ must be independent. Let $r(a_1) < r(a_2) < \ldots < r(a_x)$, where $r(e)$ denotes the rank of $e$ in the sorted $S$.

# Proof (1/3)

Claim 1. A is maximal. In other words, A is not a subset of any other independent subset in F.

# Proof (1/3)

Claim 1. A is maximal. In other words, A is not a subset of any other independent subset in F.

Proof. Suppose that $A \subset B$ for some $B \in F$ and $|B| > |A|$. Let z be some element in $B \setminus A$ so that $A \cup \{z\}$ in F. Such z exists due to the exchange property. By the hereditary property, $\{a_1, a_2, \ldots, a_t\} \cup \{z\}$ in F for any $t \in [1, x]$.

If $r(a_x) < r(z)$, then Greedy should have added some e into A so that $r(a_x) < r(e) \le r(z)$. $\rightarrow\leftarrow$

Otherwise $r(a_{t-1}) < r(z) < r(a_t)$ for some $t \in [1, x]$, then Greedy should have added some e into A so that $r(a_{t-1}) < r(e) \le r(z)$.

$\rightarrow\leftarrow$

# Proof (2/3)

Claim 2. Every maximal subset in F has the same size.

Proof. Let A and B be two maximal subsets in F and $|A| < |B|$. By the exchange property, there exists some z in $B \setminus A$ so that $A \cup \{z\}$ in F, contradicting the maximality of A.

# Proof (2/3)

Claim 2. Every maximal subset in F has the same size.

Proof. Let A and B be two maximal subsets in F and $|A| < |B|$. By the exchange property, there exists some z in $B \setminus A$ so that $A \cup \{z\}$ in F, contradicting the maximality of A.

Claim 3. Some maximal subset in F is a max-weight subset in F.

Proof. Let A be a max-weight subset but A is a proper subset of some B in F. Then, B also has the max-weight because the weight function w is non-negative. Such B is a witness.

# Proof (2/3)

Claim 2. Every maximal subset in F has the same size.

Proof. Let A and B be two maximal subsets in F and $|A| < |B|$. By the exchange property, there exists some z in $B \setminus A$ so that $A \cup \{z\}$ in F, contradicting the maximality of A.

Claim 3. Some maximal subset in F is a max-weight subset in F.

Proof. Let A be a max-weight subset but A is a proper subset of some B in F. Then, B also has the max-weight because the weight function w is non-negative. Such B is a witness.

Both A and a max-weight subset are maximal.

# Proof (3/3)

Let the max-weight subset be $B = \{b_1, b_2, \ldots, b_x\}$ where $r(b_1) < r(b_2) < \ldots < r(b_x)$. Since $A \neq B$, there exists some $t \in [1, x]$ so that $r(a_1) = r(b_1)$, $r(a_2) = r(b_2)$, …, $r(a_{t-1}) = r(b_{t-1})$, $r(a_t) < r(b_t)$. If there are multiple max-weight susbets, we pick B as the one whose t is the largest.

# Proof (3/3)

Let the max-weight subset be $B = \{b_1, b_2, \ldots, b_x\}$ where $r(b_1) < r(b_2) < \ldots < r(b_x)$. Since $A \neq B$, there exists some $t \in [1, x]$ so that $r(a_1) = r(b_1)$, $r(a_2) = r(b_2)$, …, $r(a_{t-1}) = r(b_{t-1})$, $r(a_t) < r(b_t)$. If there are multiple max-weight susbets, we pick B as the one whose t is the largest.

Theorem. A = B, and therefore A is a max-weight subset.

# Proof (3/3)

Let the max-weight subset be $B = \{b_1, b_2, \ldots, b_x\}$ where $r(b_1) < r(b_2) < \ldots < r(b_x)$. Since $A \neq B$, there exists some $t \in [1, x]$ so that $r(a_1) = r(b_1)$, $r(a_2) = r(b_2)$, …, $r(a_{t-1}) = r(b_{t-1})$, $r(a_t) < r(b_t)$. If there are multiple max-weight susbets, we pick B as the one whose t is the largest.

Theorem. A = B, and therefore A is a max-weight subset.

Proof. Let $A' = \{a_1, a_2, \ldots, a_t\}$. By exchange property, we could iteratively add some element from $B \setminus A'$ to $A'$ until $A'$ has the same size as B. Finally, $A'$ becomes $B \cup \{a_t\} \setminus \{z\}$ for some z in $\{b_t, b_{t+1}, \ldots, b_x\}$. Since $r(a_t) < r(b_t) \leq r(z)$, $w(a_t) \geq w(z)$ and A' is another max-weight subset, violating the condition that B is the one whose t is the largest.

# Submodularity

# Submodular functions

Let S be a set of n elements. Let f be a function that maps any subset of S to a real number.

We say such f is **submodular** if

$$f(X \cup \{e\}) - f(X) \geq f(Y \cup \{e\}) - f(Y)$$

for any $X \subseteq Y \subseteq S$.

# Monotone functions

Let S be a set of n elements. Let f be a function that maps any subset of S to a real number.

We say such f is **monotone** if

$$f(X) \leq f(Y)$$

for any $X \subseteq Y \subseteq S$.

# Submodular Maximization

Given a set S, a submodular function f: $2^S \rightarrow R$, and an integer k, find a subset X of S so that:

(1) $|X| \le k$
(2) f(X) is maximum among all subsets of S that has cardinality $\le$ k.

# Submodular Maximization

Given a set S, a submodular function f: $2^S \rightarrow R$, and an integer k, find a subset X of S so that:

(1) $|X| \leq k$
(2) f(X) is maximum among all subsets of S that has cardinality $\leq$ k.

Thm. If S is submodular, monotone, and non-negative, then there exists a greedy algorithm that can find a (1-1/e)-approximation.

# The greedy algorithm

```
Greedy(S, f, k){

    A = ∅; // let the answer be an empty set initially
    for(i=1; i≤k; ++i){
        for(j=1; j≤|S|; ++j){
            if(f(A ∪ {s_j}) ≥ f(A ∪ {s_k}) for all k's){
                A = A ∪ {s_j};
            }
        }
    }
    return A;
}
```

# The maximum coverage problem

Given n points. Place k circles, each of which has radius d and is centered at some given point, so that the number of covered points is maximized.