COMP20003 Workshop Week 7

1	Sorting, Insertion Sort and Selection Sort
2	Quicksort
3	MST = ?
4	Ass2 Q&A
	MST in ?? Week 8
	How far did you go with ass2: Send me a letter: A- done B- done stages 1+2, doing stage 3 C- just finished stages 1+2 D- finished stage 1 E- none of the above

Sorting Algorithms

Any problem with:

- Selection Sort?
- Insertion Sort?
- Quick Sort?

Properties of sorting algorithms:

- *in-place: not using additional arrays to store data*
- *stable: maintaining the relative order of data with equal keys*

Note: *In this workshop, suppose we need to sort an array in non-decreasing order.*

Selection Sort (choosing the smallest)

-	5_1 9 4 5_2 1 2 7 n
1	1 | 9 4 5_2 5_1 2 7 n-1
2	1 2 | 4 52 51 9 7 n-2
3	1 2 4 | 52 51 9 7
4	1 2 4 52 | 51 9 7
5	1 2 4 52 51 | 9 7
6	1 2 4 52 51 7 | 9
	= n+ (n-1) ++…1 =
	Theta(n^2) = n(n+1)/2

- Run the algorithm on the input array: [5 9 4 5 1 2 7]
- What is the time complexity of the algorithm? : θ(n^2)

  Best case: Average :
- Is the sorting algorithm stable? Y/N
- Is the algorithm in-place?

Selection Sort (choosing the smallest): Check your answer

-	| 5_1 9 4 5_2 **1** 2 7
1	1 | 9 4 5_2 5_1 **2** 7
2	1 2 | **4** 5_2 5_1 9 7
3	1 2 4 | **5_2** 5_1 9 7
4	1 2 4 5_2 | **5_1** 9 7
5	1 2 4 5_2 5_1 | 9 **7**
6	1 2 4 5_2 5_1 7 | 9

- Run the algorithm on the input array: [5 9 4 5 1 2 7]
- What is the time complexity of the algorithm? : $\boldsymbol{\theta}(n^2)$

- Is the sorting algorithm stable? No
- Is the algorithm in-place? Yes

Insertion Sort

	5_1 | 9 4 5_2 1 2 7
1	1 2 4 5 _5 7 9
2	
3	
4	
5	
6	

- Run the algorithm on the input array: [5 9 4 5 1 2 7]
- What is the time complexity of the algorithm? : O( ) $\boldsymbol{\theta}$( )

  Best case: Average :
- Is the sorting algorithm stable?
- Is the algorithm in-place?

Insertion Sort – check your answer

	5_1 | **9** 4 5_2 1 2 7
1	5_1 9 | **4** 5_2 1 2 7
2	4 5_1 9 | **5_2** 1 2 7
3	4 5_1 5_2 9 | **1** 2 7
4	1 4 5_1 5_2 9 | **2** 7
5	1 2 4 5_1 5_2 9 | **7**
6	1 2 4 5_1 5_2 7 9 |

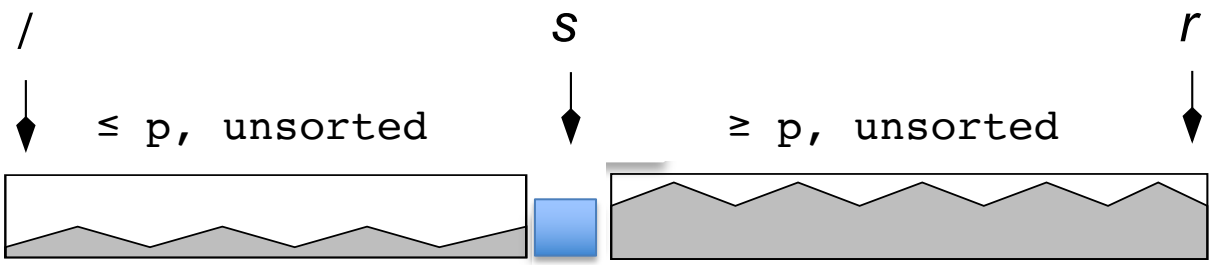
- Run the algorithm on the input array: [5 9 4 5 1 2 7]
- What is the time complexity of the algorithm? : O(n^2)

  Best case: O(n) Average : O(n^2)
- Is the sorting algorithm stable? Yes
- Is the algorithm in-place? Yes

Quicksort (usage: `Quicksort(A[0..n-1])`

```
function QUICKSORT(A[l..r])
    if l < r then
        s ← PARTITION(A[l..r])
        QUICKSORT(A[l..s − 1])
        QUICKSORT(A[s + 1..r])
```

p = A[s] is called the *pivot* of this partitioning

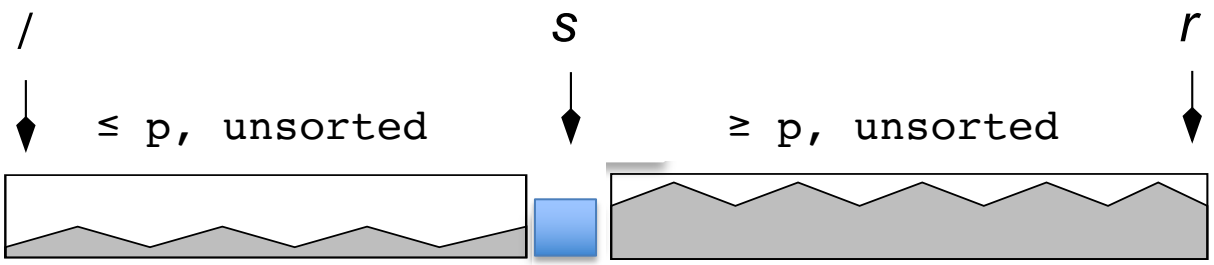


Q: Suppose that a PARTITION of n elements has the complexity of $\theta(n)$
What is the complexity of QUICKSORT?

Quicksort: check your answer on complexity

```
function QUICKSORT(A[l..r])
    if l < r then
        s ← PARTITION(A[l..r])
        QUICKSORT(A[l..s − 1])
        QUICKSORT(A[s + 1..r])
```

p = A[s] is called the *pivot* of this partitioning



What is the complexity of QUICKSORT?

BEST/AVERAGE O(n logn)

```
--------------  n
-------  -------  2 x (n/2)
---  ---  ---  ---  n
-  -  -  -  -  -  -  -  n
```

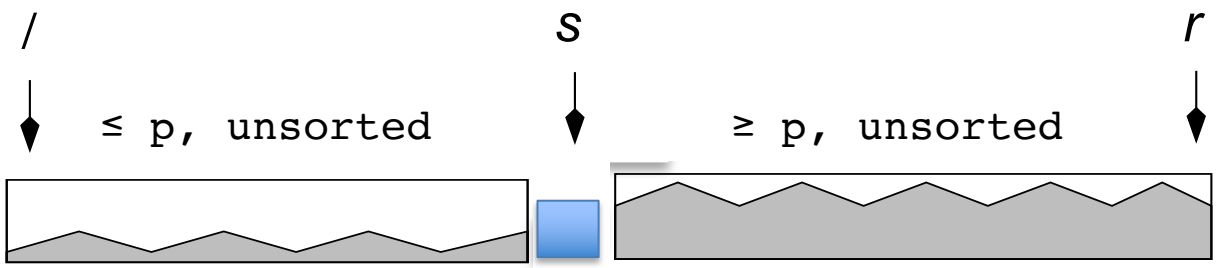
WORST $O(n^2)$

```
--------------  n
 -------------  n-1
  ------------  n-2
   -----------  n-3
        ...
```

Quicksort (usage: `Quicksort(A[0..n-1])`

```
function QUICKSORT(A[l..r])
    if l < r then
        s ← PARTITION(A[l..r])
        QUICKSORT(A[l..s − 1])
        QUICKSORT(A[s + 1..r])
```

p = A[s] is called the *pivot* of the partitioning



Q: How to do the Partitioning?

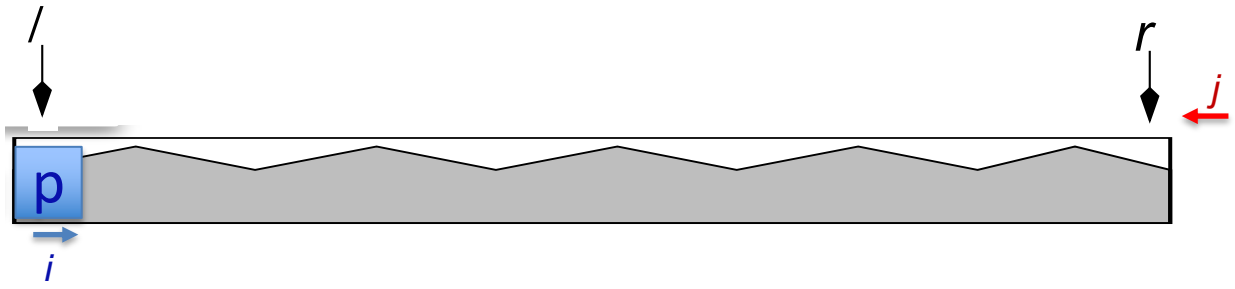
Choose p= any element in A, say, p= A[l]

Partition A[l+1..r]

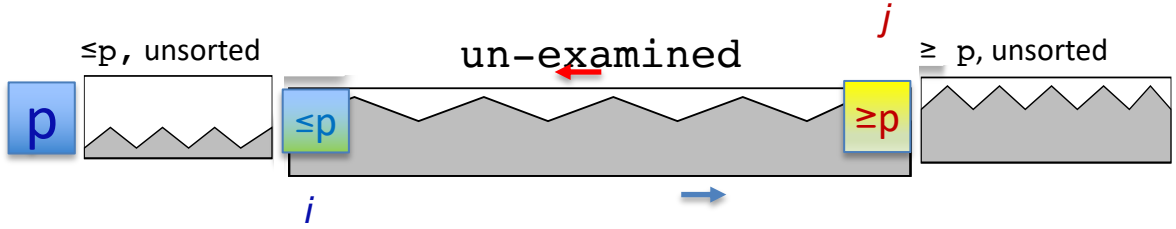
Swap A[l] with the last of the ≤ p *half*

Quicksort: Hoare's Partitioning

set pivot p = A[l] and leave it untouched for a while
set lower pointer i and higher pointer j

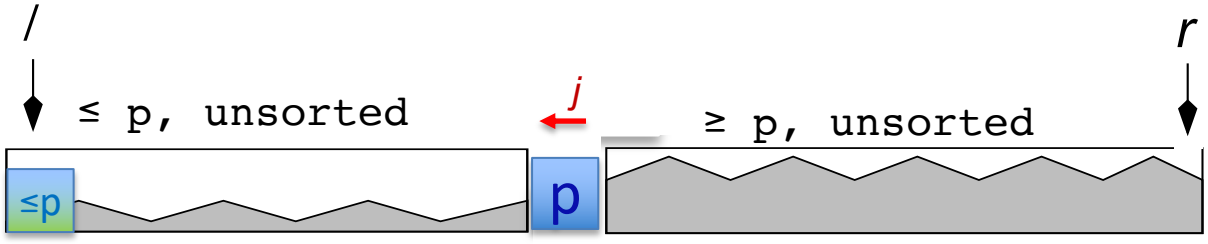


loop: + move i forward and j backward, stopping at ≥p and ≤p respectively



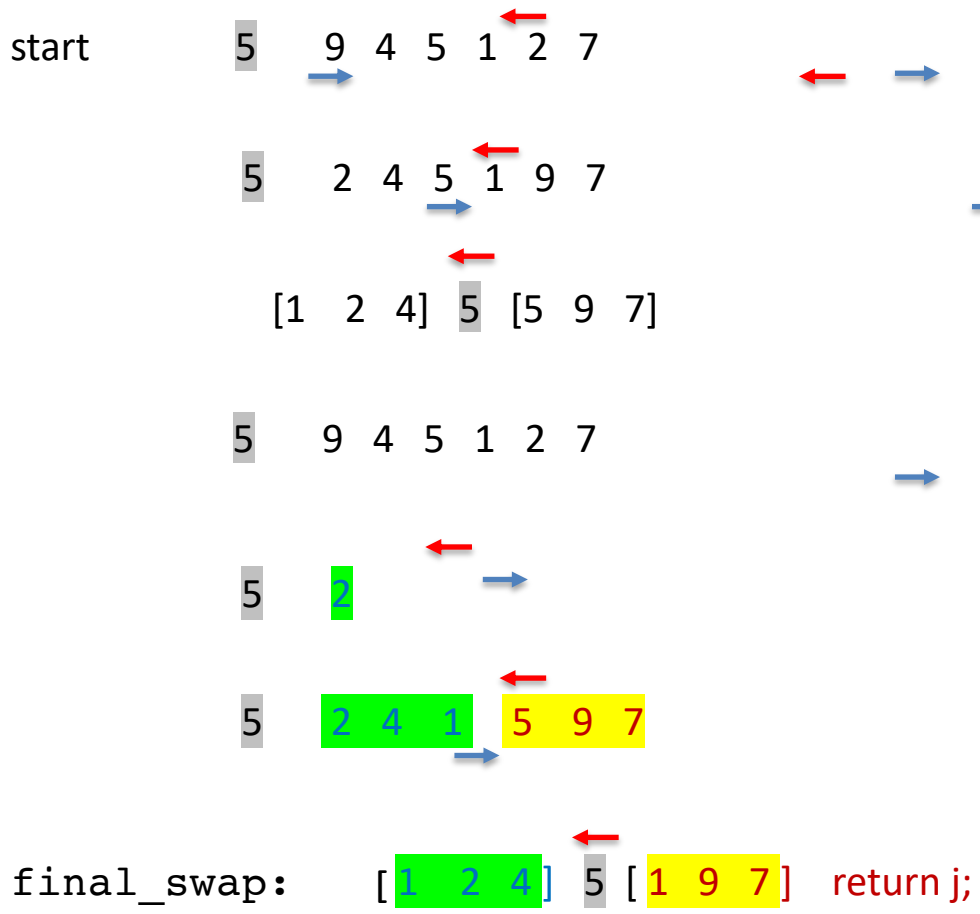
+ if i<j, swap A[i] and A[j] and continue the loop.

swap_pivot: at the end: swap pivot with A[j], return j



Partition (=leftmost)

1. set p= A[l] (=leftmost), keep A[l] untouched
2. Start with i= l, j=r+1
3. move: + Move i forward, stop when A[i]>=p
   + Move j backward, stop when A[j]<=p
4. swap: If i<j: swap A[i], A[j] then go back to step 3
5. final_swap: Swap A[l] and A[j], return m= j



+ Move i forward, stop when A[i]>=p:

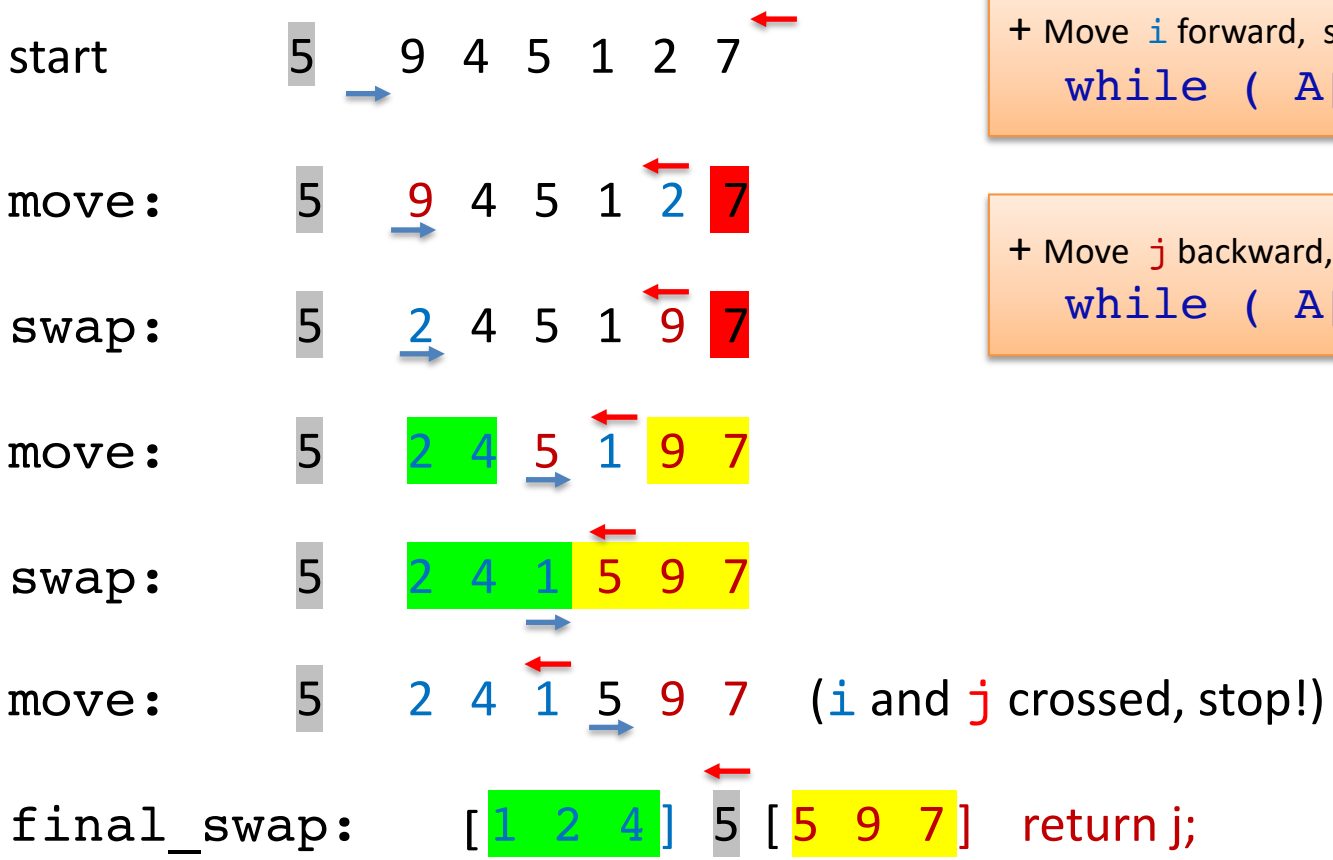
```
while ( A[++i] < p);
```

+ Move j backward, stop when A[j]>=p:

```
while ( A[--j] > p);
```

Partition

1. set p= A[l] (=leftmost), keep A[l] away
2. Start with i= l, j=r+1
3. move: + Move i forward, stop when A[i]>=p
   + Move j backward, stop when A[j]<=p
4. swap: If i<j: swap A[i], A[j] then go back to step 3
5. final_swap: Swap A[l] and A[j], return m= j



+ Move i forward, stop when A[i]>=p:

```
while ( A[++i] < p);
```

+ Move j backward, stop when A[j]>=p:

```
while ( A[--j] > p);
```

Your Task: Finish the quicksort algorithm

Initial array: (2) 9 4 5 1 5 7

after 1st partition: [1 2 4] 5 [5 9 7]

[(1) 2 4] 5 [(5) 9 7]

→ 1 [(2) 4] 5 5 [(9) 7]

1 2 [4] 5 5 [7] 9

Finger Exercise: Text me next rows, using () to surround the pivot, [] to surround the LHS & RHS, as in: (5) 945127 → [124] 5 [597]

Discussion:

- What's the complexity of partitioning on n elements?
- What if I want to use some other element as pivot, say A[k]?
- What is a good choice for k (ie. pivot)?

Your Task: Check your answer

Initial array: (5) 9 4 5 1 2 7

after 1st partition: [1 2 4] 5 [5 9 7]

[(1) 2 4] 5 [(5) 9 7]

1 [2 4] 5 5 [9 7]

1 [(2) 4] 5 5 [(9) 7]

1 2 [4] 5 5 [7] 9

1 2 4 5 7 9

Discussion:

- What's the complexity of partitioning on n elements? $\theta(n)$
- What if I want to use some other element as pivot, say A[k]?

  → just swap A[k] and A[l], then continue the algorithm as it is
- What is a good choice for k (ie. pivot)?

  → The Lord of Randomness has a great power!

Other Materials

- Lectures!
- Grady’s Workshop Slides in Lesson
- Text book and etc

Quiz 1

The best case of Selection sort is:

a. O(n log n).

b. O(n).

c. $O(n^2)$.

d. O(log n).

When?

Quiz 2

The best case of Insertion Sort is:

a. O(n log n).

b. O(n).

c. $O(n^2)$.

d. O(log n).

When?

Quiz 3

The average case of Insertion Sort is:

a. O(n log n).

b. O(n).

c. $O(n^2)$.

d. O(log n).

Quiz 4

The average case of Quick Sort is:

a. O(n log n).

b. O(n).

c. $O(n^2)$.

d. O(log n).

Quiz 5

The big-O complexity of Quick Sort is:

a. O(n log n).

b. O(n).

c. $O(n^2)$.

d. O(log n).

Sorting algorithms

	Selection Sort	Insertion Sort	Quick Sort	Merge
Basic Idea	for (i=0; i<n-1; i++): • find the smallest of A[i..n-1] • swap with A[i] • hence done with A[0..i]	for (i=1; i<n; i++) : • insert A[i] to the *sorted* A[0..i-1] to make A[0..i] sorted	Choose a pivot, partition array into a *lesser* and a *greater* (than pivot) halves. Do recursively with each half.	Split …
Complexity	$\theta(n^2)$	$O(n^2)$	$O(n^2)$	
Best case	$O(n^2)$	$O(n)$	$O(n \log n)$	
Average	$O(n^2)$	$O(n^2)$	$O(n \log n)$	
In-place?	✓	✓	✓	
Stable?	✗	✓	✗	

Sorting Algorithms

	Selection	Insertion	Quick	Merge
Basic Idea (for one iteration)	Identify the smallest and swap it with the first.	Insert next element to the *sorted* LHS to make the new, extended LHS sorted	Partition array into a *lesser* and a *greater* (than pivot) halves.	
Complexity	$\Theta(n^2)$	$O(n^2)$	$O(n^2)$	
In-place?	✓	✓	✓	
Stable?	✗	✓	✗	
What's Good?	• minimal number of swaps: O(n) • useful when swaps are expensive	• stable • Fast for nearly sorted arrays, or for small-size array	• O(n logn) on average	

from BST to MST ?

2015/2017/2019 MST

- Sample MST on Canvas
  - New sample MST
  - 2015 MST
    - Warning: a bit harder wrt. the complexity question!
  - 2017 MST
  - 2019 MST

LAB TIME

MAIN ROOM

- Assignment 2:
  - How to do incremental voronoi
  - Other problems
- Questions from past MST
  - Complexity
  - Arrays, Linked Lists, Stacks?, Queues?, BST, AVL and Rotation, Hashing
  - Sorting, incl. quicksort, but probably excluding mergesort (ask Nir)
  - Programing: dynamic arrays, strings, linked lists and ?

BREAK-OUT ROOMS

- Programing 6.1: Partitioning
  - You have a small base code, just fill in a function for partitioning ☺
- Group:
  - Share screen on pdf and work through mid-sem
- Individual: Do/Finish Assignment 2

Assignment 2

How far did you go?

A- finished, program works, but not sure if stage 3 is correct

B- finished stages 1 and 2, no problem with stage 4, but struggling with stage 3

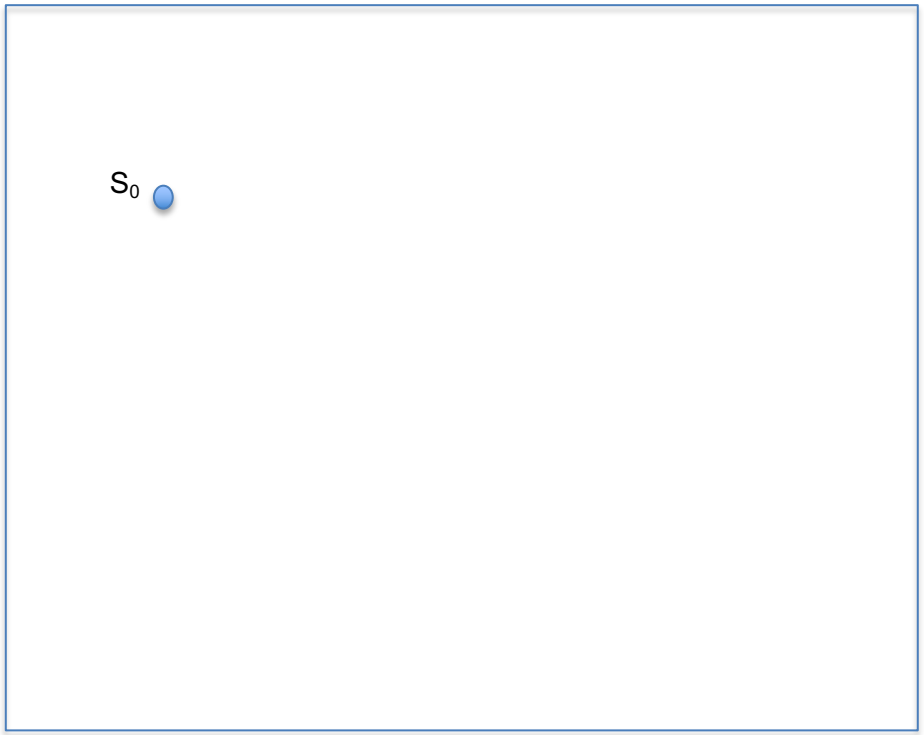
C- finished stage 1+2, struggling with stage 3, don’t know how to do stage 4

D- just finished stage 1

E- none of the above

Assignment 2 Q&A

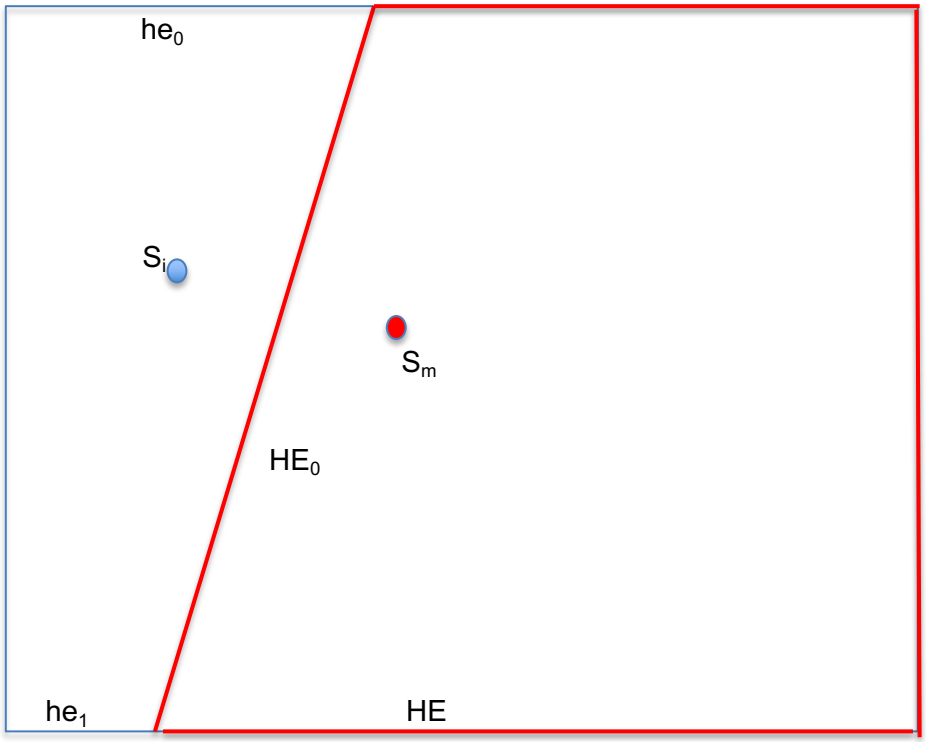
Incremental voronoi cell update



- Build the first face from the polygon.
- Add the first point (tower), the first face become the first voronoi cell (VC).

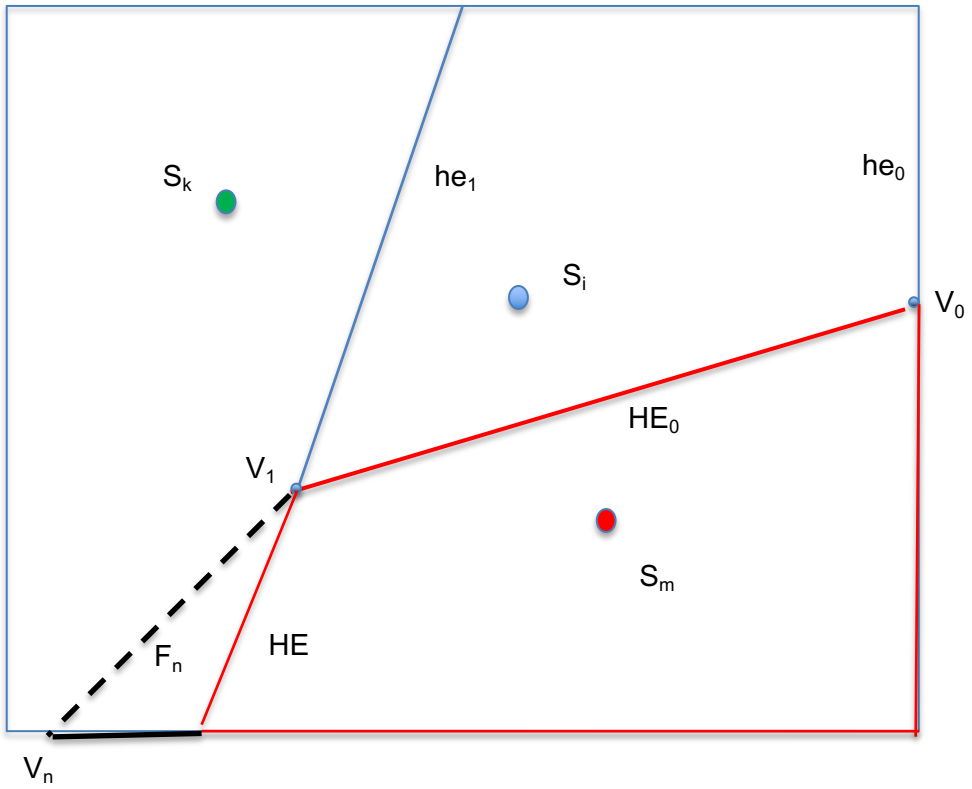
- face: which tower_t *t
- tower_t : int face

Incremental voronoi cell update



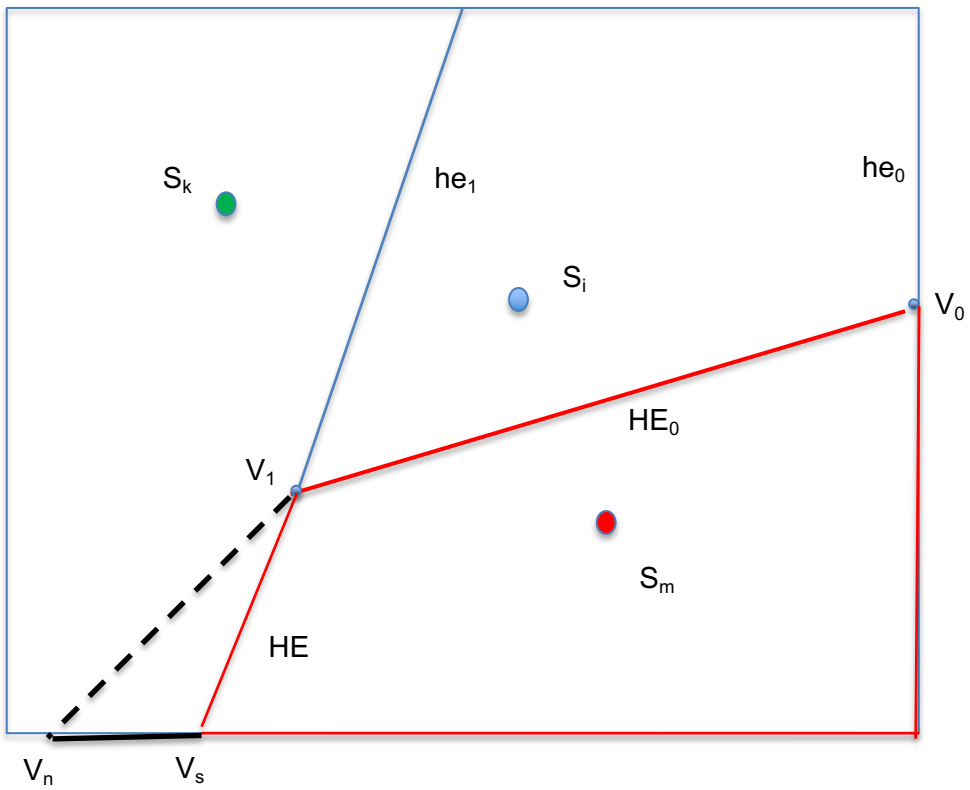
- For each subsequent point S_m:
  - Find the Voronoi cell $VC(S_i)$ that contains S_m.
  - Compute the bisector b_{im} of S_i and S_m.
  - Use the bisector to build the split from, say, halfEdge (HE) he_0 to he_1 in clockwise direction of VC(S_i) [it's counter-clockwise wrt. point S_m]
  - Do the split, get a new face, make it be the new cell VC(S_m)
  - **If he_1 is non-voronoi edges: the algorithm terminates.** Otherwise: …

Incremental voronoi cell update



- Otherwise: suppose that HE_0 is the halfedge from V_1 to V_0 in $VC(S_m)$ along the split line. Examine the next edge of $VC(S_m)$ in the counter-clockwise manner. Let's call it `HE`.
- **If HE is a voronoi edge:**
  - Find the cell/face $VC(S_k)$ that is adjacent to
  - build intersection b_{mk} and use that to split $VC(S_k)$. Suppose that the bisector intersects another edge at point V_n. Note that we have a new *face* F_n that is surrounded by black edges and one red edge.
  - We need to ***join*** the face F_n to the cell $VC(S_m)$

Joining a face to VC:
Case 1: when the two intersected edges are adjacent



Here:

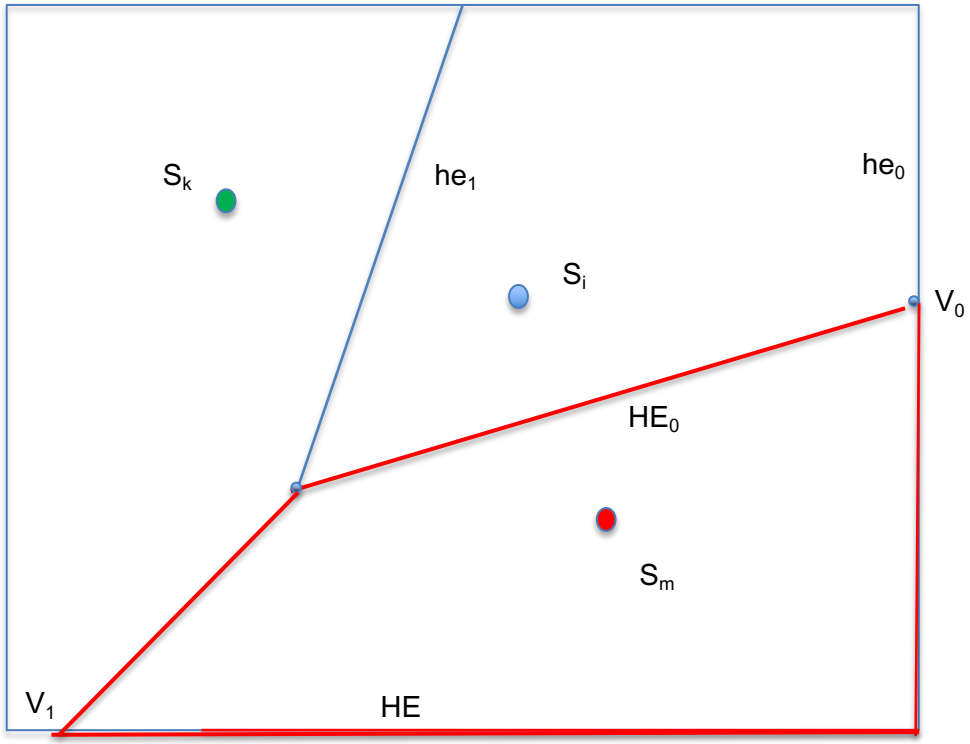
- note that Vs is the only vertex that is shared between VC(Sm) and the new face Fn
- Two intersected edges are HE and Vs→Vn. Here the second one is non-voronoi

How to join Fn to VC(Sm)?:

delete the shared vertex

delete the edge shared between VS(Sm) and Fn

Joining a face to VC:
Case 1: when the two intersected edges are adjacent

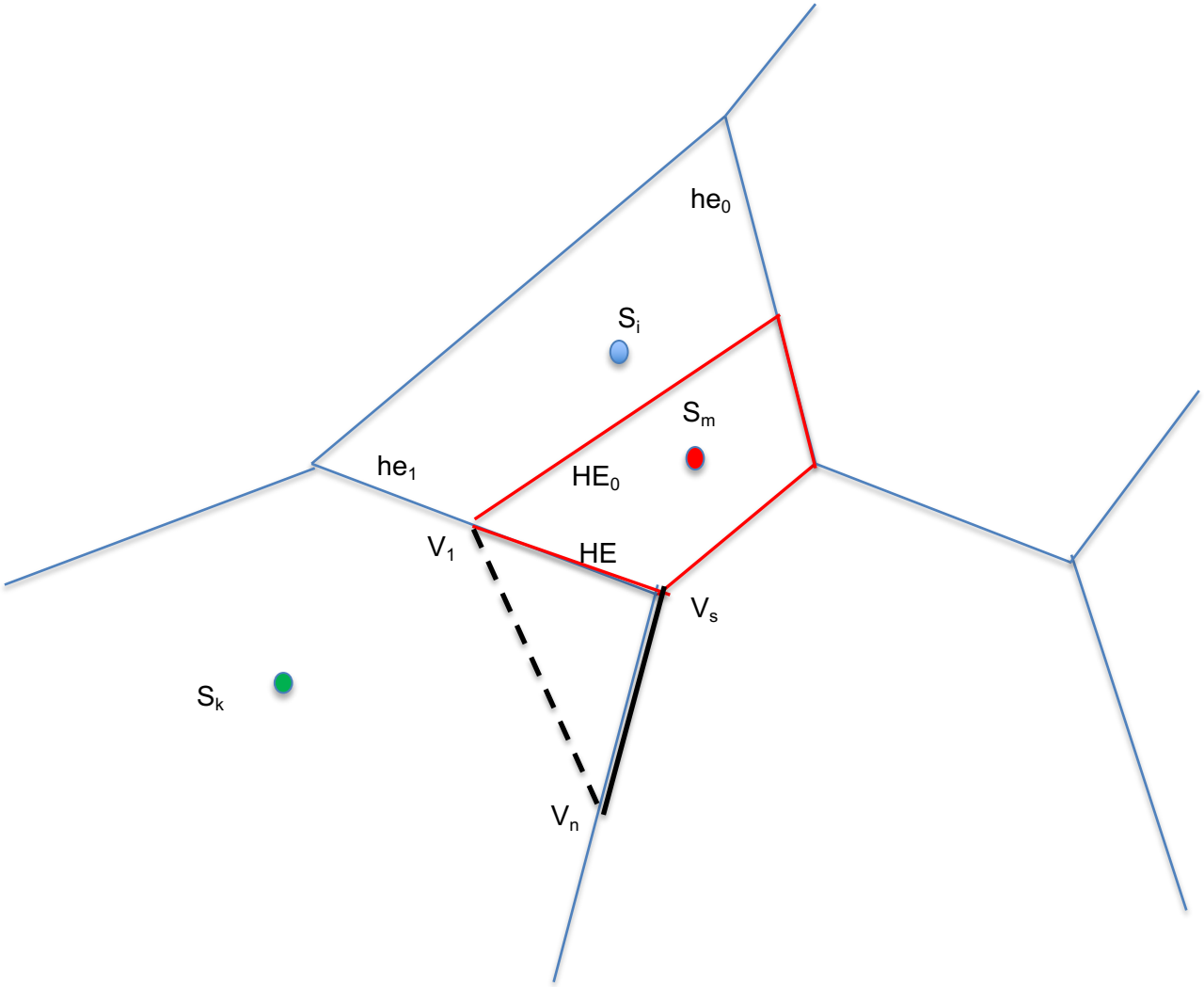


After joining.

After joining, we move HE further to HE->prev.

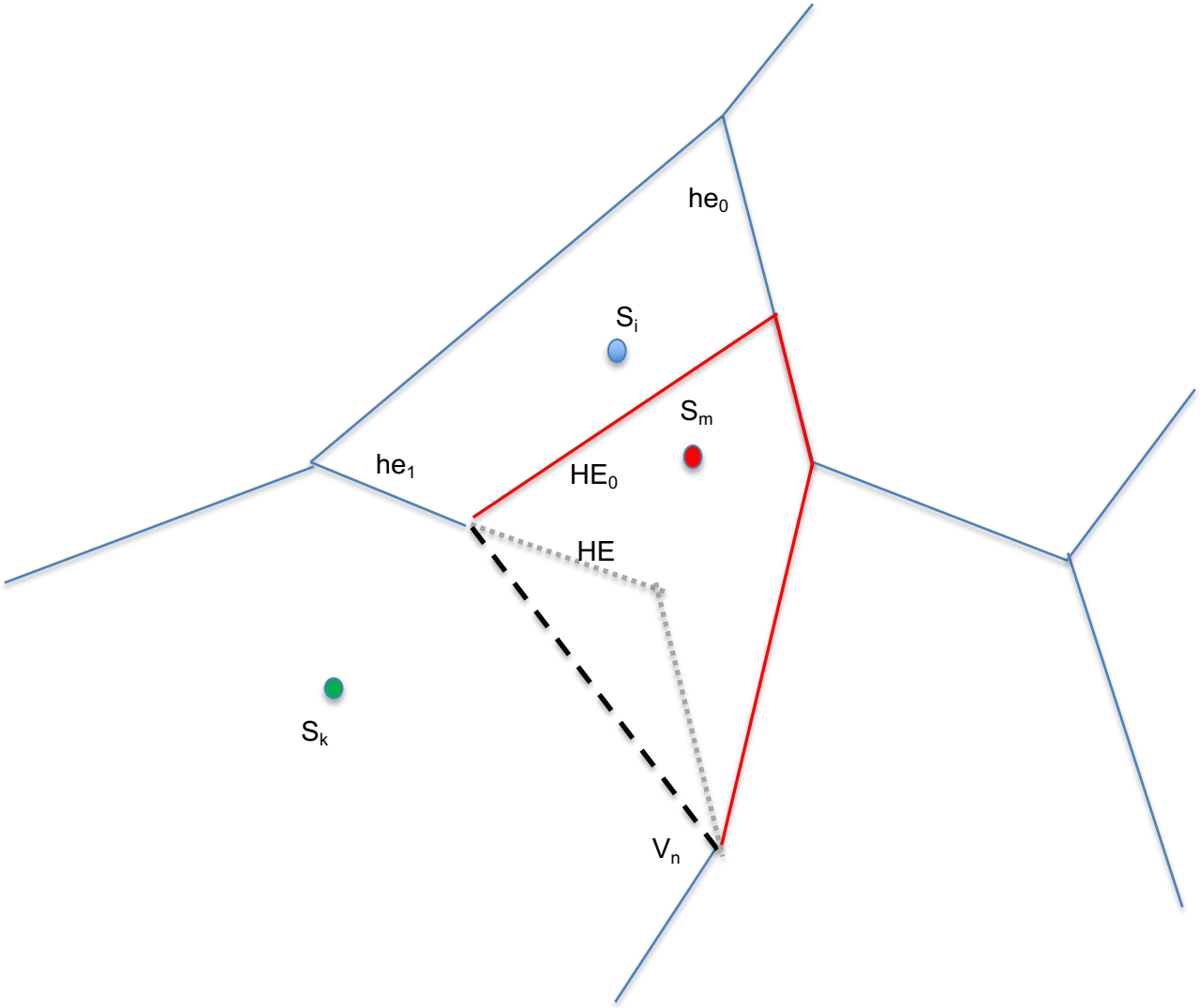
Note that in this case, the new HE is a non-Voronoi, and the algorithm terminates.

Joining a face to VC:
Case 1: when the two intersected edges are adjacent



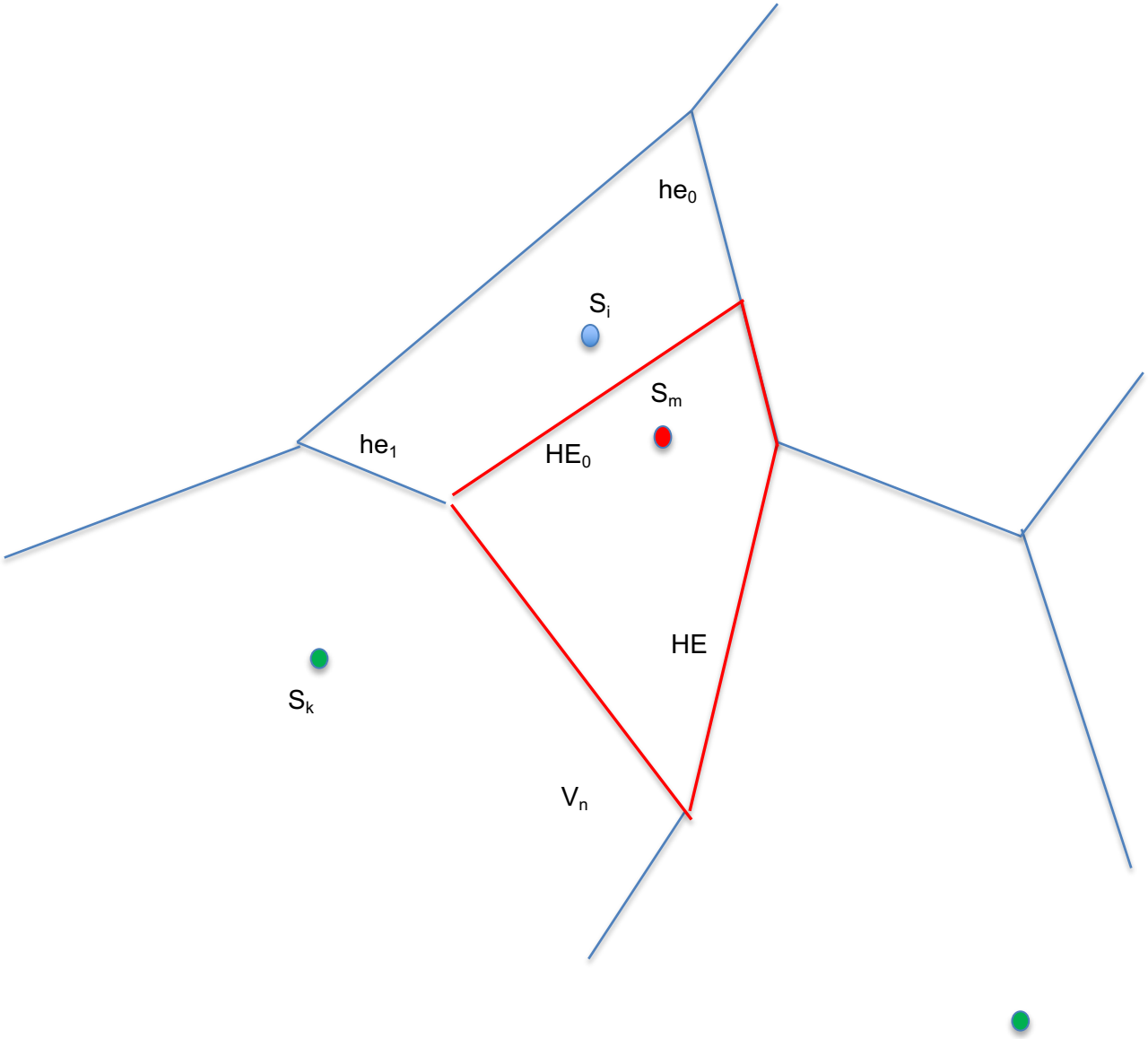
Here is another situation of case 1, but the procedure is the same.

Case 1:



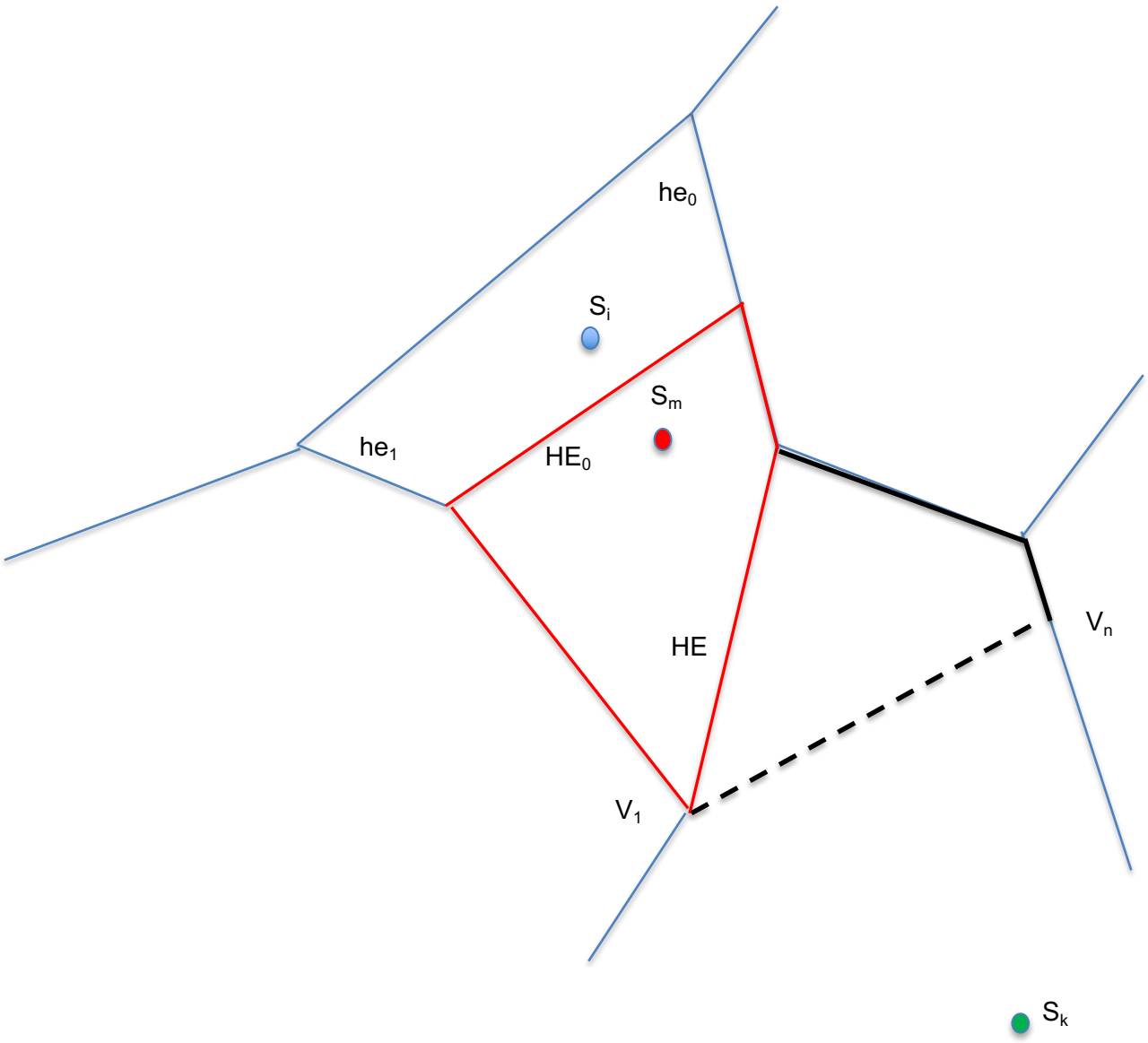
V1 has been deleted, and the dotted grey edges are also deleted.

Case 1: after joining



Advance HE to HE->prev

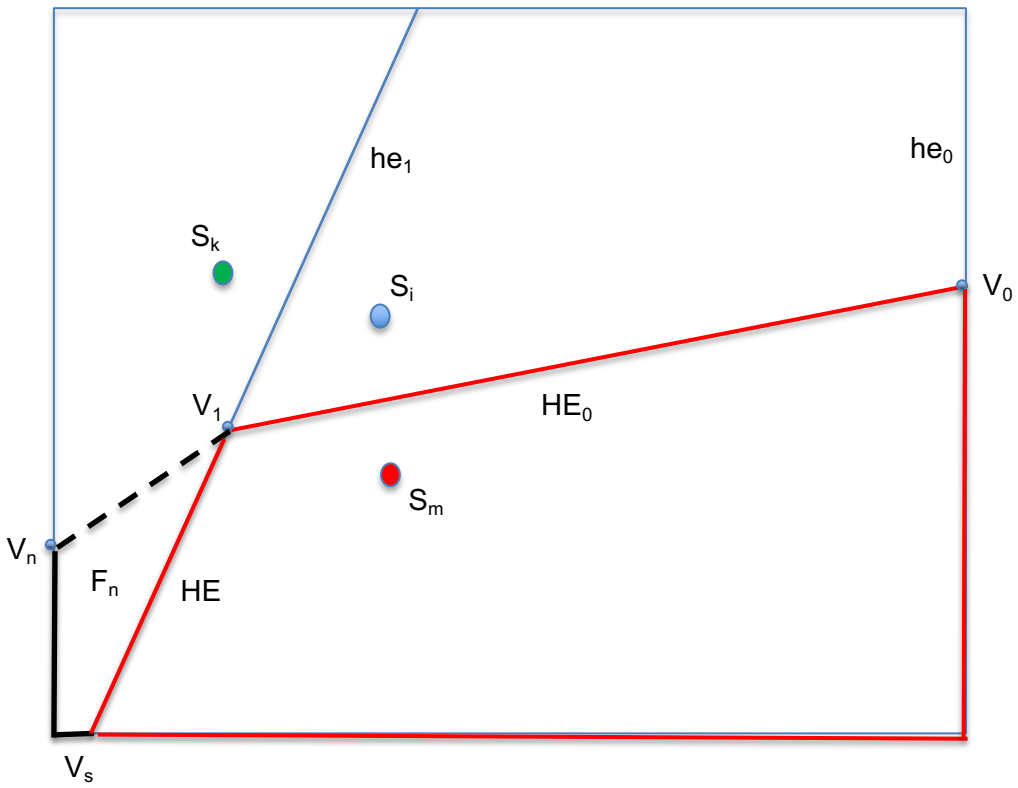
Joining a face to VC:
Case 2: The intersected edges are not adjacent



Continue the work by using bisector between the red and the green points.

Now we have case 2 of joining faces.

Case 2: Another situation



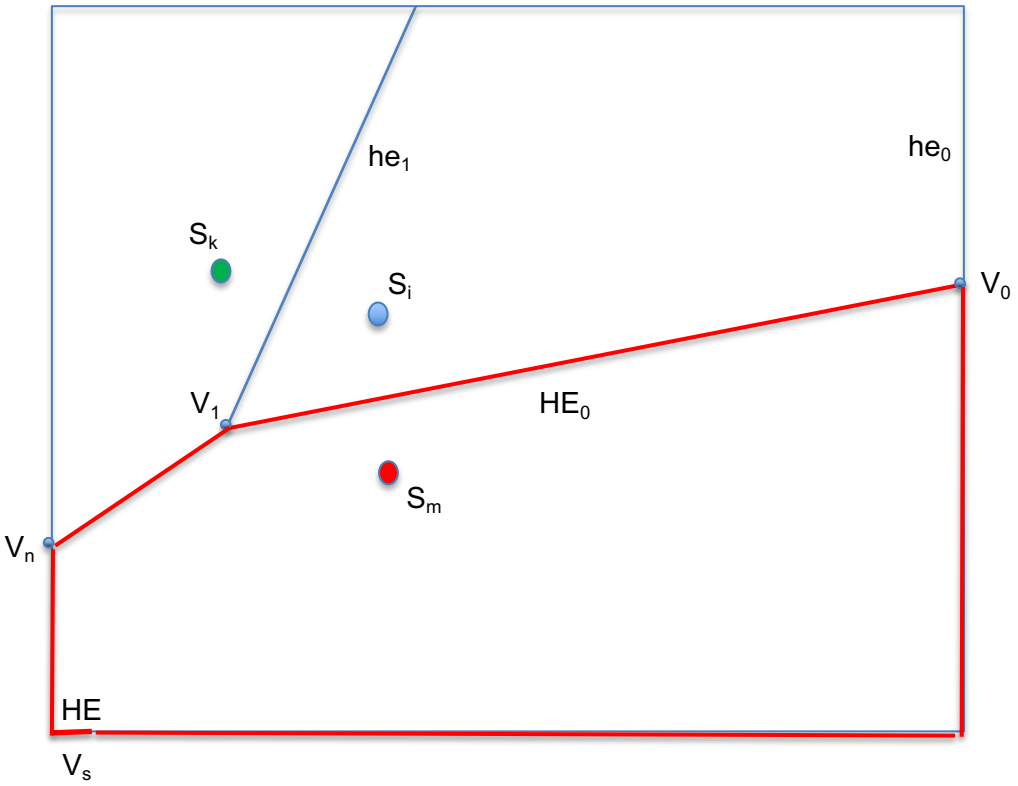
Here is another situation of Case 2.

How join Fn to VC(Sm):

- Join all the black edges, starting from Vn→V1, into VC(Sm). Stop when reaching Vs
- Delete the shared edge V1→Vs

Note: I believe that this procedure is the same as described in spec, just in a different order of edge processing.

Case 2: Another situation



After joining.

Move HE to HE->prev. Note that here:

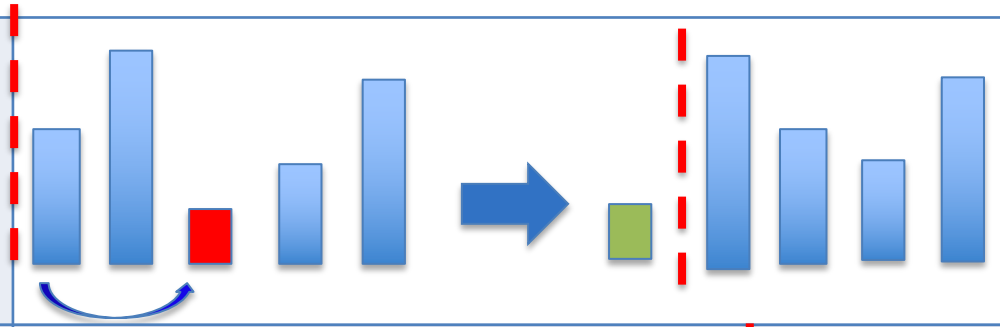
- the new HE is non-voronoi, hence the algorithm terminates
- the new HE here is only a part of an edge of the original region. We can join this HE to the edge, but we don't have to

- The other situation of Case 2 is similar to deal with, but we need to continue after that

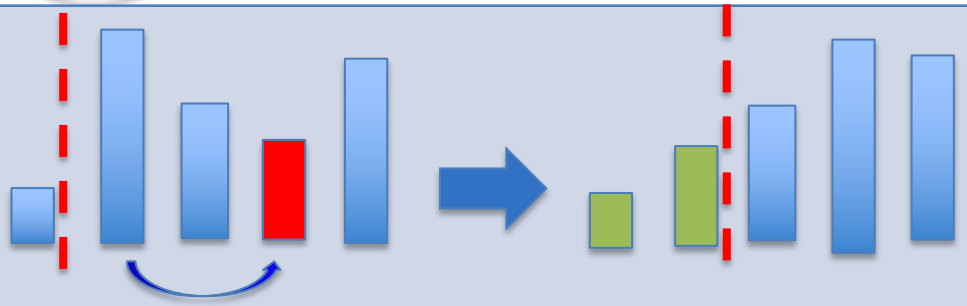
Additional Materials

Selection Sort: n=5, (here: selecting the smallest)

Round 0: consider `A[0..4]`
- *determine position of the smallest*
- *swap with the first, ie.* `A[0]`

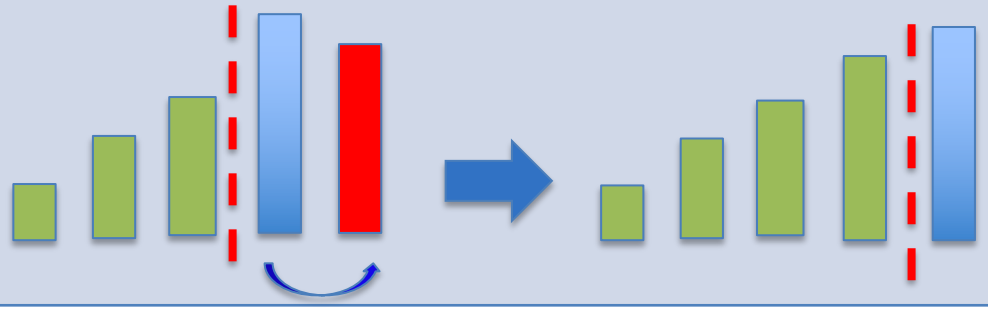


Round i: consider `A[i..n-1]`
- *determine position of the smallest*
- *swap with the first, ie.* `A[i]`



Round n-2: consider `A[n-2..n-1]`
- *determine position of the smallest*
- *swap with the first, ie.* `A[n-2]`

Done!



Selection Sort

-	A N A L Y S I S
1	
2	
3	
4	
5	
6	
7	

i. Run the algorithm on the input array: [A N A L Y S I S]

ii. What is the time complexity of the algorithm?

iii. Is the sorting algorithm stable?

iv. Is the algorithm in-place?

Selection Sort

-	**A** N A L Y S I S
1	A | **N** **A** L Y S I S
2	A A | **N** L Y S **I** S
3	A A I | **L** Y S N S
4	A A I L | **Y** S **N** S
5	A A I L N | **S** Y S
6	A A I L N S | **Y** **S**
7	A A I L N S S | Y

i. Run the algorithm on the input array: `[A N A L Y S I S]`

ii. What is the time complexity of the algorithm?

iii. Is the sorting algorithm stable?

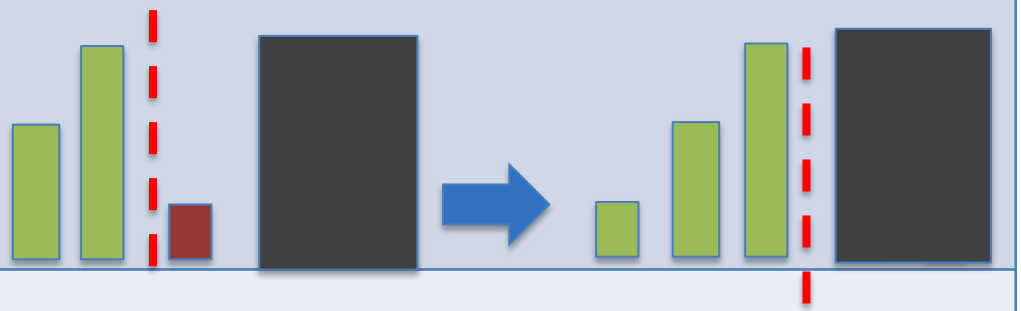
iv. Is the algorithm in-place?

Insertion Sort: understanding (n=5)

Round 1: consider `A[1]`

- `A[0..0]` *is sorted*
- *insert* `A[1]` *to the left so that* `A[0..1]` *is sorted*

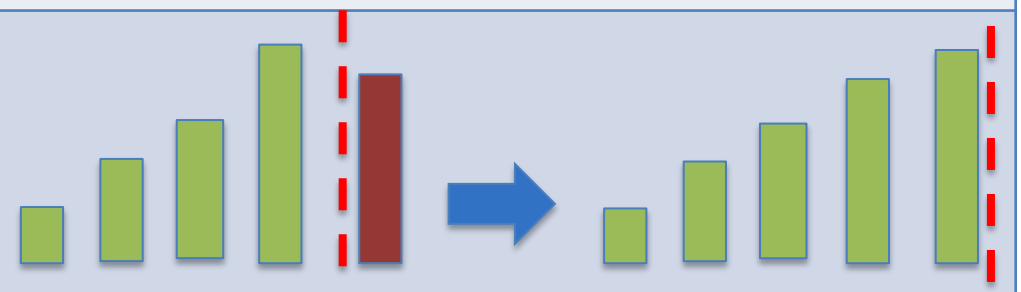
Round 2:



Round i: consider `A[i]`

- `A[0..i-1]` *is sorted*
- *insert* `A[i]` *to the left so that* `A[0..i]` *is sorted*

Round n-1: consider `A[n-1]`



Insertion Sort

	A N A L Y S I S
1	
2	
3	
4	
5	
6	
7	

i. Run the algorithm on the input array: [A N A L Y S I S]
ii. What is the time complexity of the algorithm?

iii. Is the sorting algorithm stable?
iv. Does the algorithm sort in-place?

Insertion Sort

	A N A L Y S I S
1	A N A L Y S I S
2	A A N L Y S I S
3	A A L N Y S I S
4	A A L N Y S I S
5	A A L N S Y I S
6	A A I L N S Y S
7	A A I L N S S Y

i. Run the algorithm on the input array: [A N A L Y S I S]

ii. What is the time complexity of the algorithm?

iii. Is the sorting algorithm stable?

iv. Does the algorithm sort in-place?

Assignment 2 – Stage 3

The bisector b_{im} intersects two edges of VC(S_i), e_{i0} and e_{i1} in clockwise direction (ie. counter-clockwise wrt. the point S_m).

Make the split. The new resulted face is the cell VC(Sm) with HE from ei1 to ei0.

If both edges are non-Voronoi edges, i.e., don't have a twin (opposite, pair) edge then the algorithm stops, VC(S_m) is the new cell. Otherwise:

Store the new Voronoi edge of VC(S_m)*VC*(*Sm*) that connects e_{i_0}*ei*0 and e_{i_1}*ei*1 at their intersection points.

Process the HE of VC(Sm) in counter-clockwise manner, starting from vertex ei1 of the HE10 (ei1→ei0)

- If e1 is a non-voronoi HE: continue with the previous HE of HE10. Otherwise

Find the cell VC(Sk) adjacent to ei1, compute the second edge of *Si*1 that intersects *bi*1*m*, say e_{i_2}*ei*2.

If the next edge is also a Voronoi edge, retrieve the Voronoi site using the DCEL of the edge e_{i_2}*ei*2, say S_{i_2}*Si*2. If every encountered edge is a Voronoi edge, repeat the algorithm until e_{i_j}=e_{i_0}*eij*=*ei*0.

If the algorithm intersects at any stage an edge that is not a Voronoi edge, i.e., an edge of the enclosing polygon, it terminates this search for further Voronoi edges. *Note that could happen even at the beginning and we will have only Voronoi vertex because the Voronoi cell would be unbounded if we did not assume an initial polygon.*

Instead, the algorithm may intersect a non-Voronoi edge and continue its search along the initial polygon and terminates its search until it visits e_{i_0}*ei*0 again.