Week 6 Part 2

Kyle Dewey

Overview

- Array application: sorting
- Basic structs
- TA Evaluations

Sorting

Sorting

- Given some list of items in any given order, put them in sorted order
    - List of numbers ordered by <=
    - List of words in lexicographic order
    - List of plane flights in order by cost

Sorting

- 5, 3, 7, 2: (by <=)
  - 2, 3, 5, 7
- 5, 3, 7, 2: (by >=)
  - 7, 5, 3, 2
- “moo”, “cow”, “bull” (lexicographic)
  - “bull”, “cow”, “moo”

Sorting Hard Drives

Maker	Capacity	Price	Rating	Warranty
Seagate	500 GB	$80	4 / 5	3 years
Seagate	500 GB	$150	5 / 5	5 years
Hitachi	750 GB	$75	2 / 5	1 year

The Point

- The same items can have different ways of being compared

Thought Exercise

- Given a bunch of integers, devise **3** unique ways to sort them by <= (no code!)
    - If they end up being sorted in the end, the method is valid

6	2	4	1	0	9	7

Relevance to Us

- A lot of work has gone into sorting things
- Wikipedia page on sorting: 33 different methods
  - There are more
- Some generally more efficient than others, some more efficient given data that looks a certain way (such as “nearly sorted”)

Relevance to C

- There is a good chunk of code that goes into even the simpler ones
- Usually involve lots of array operations

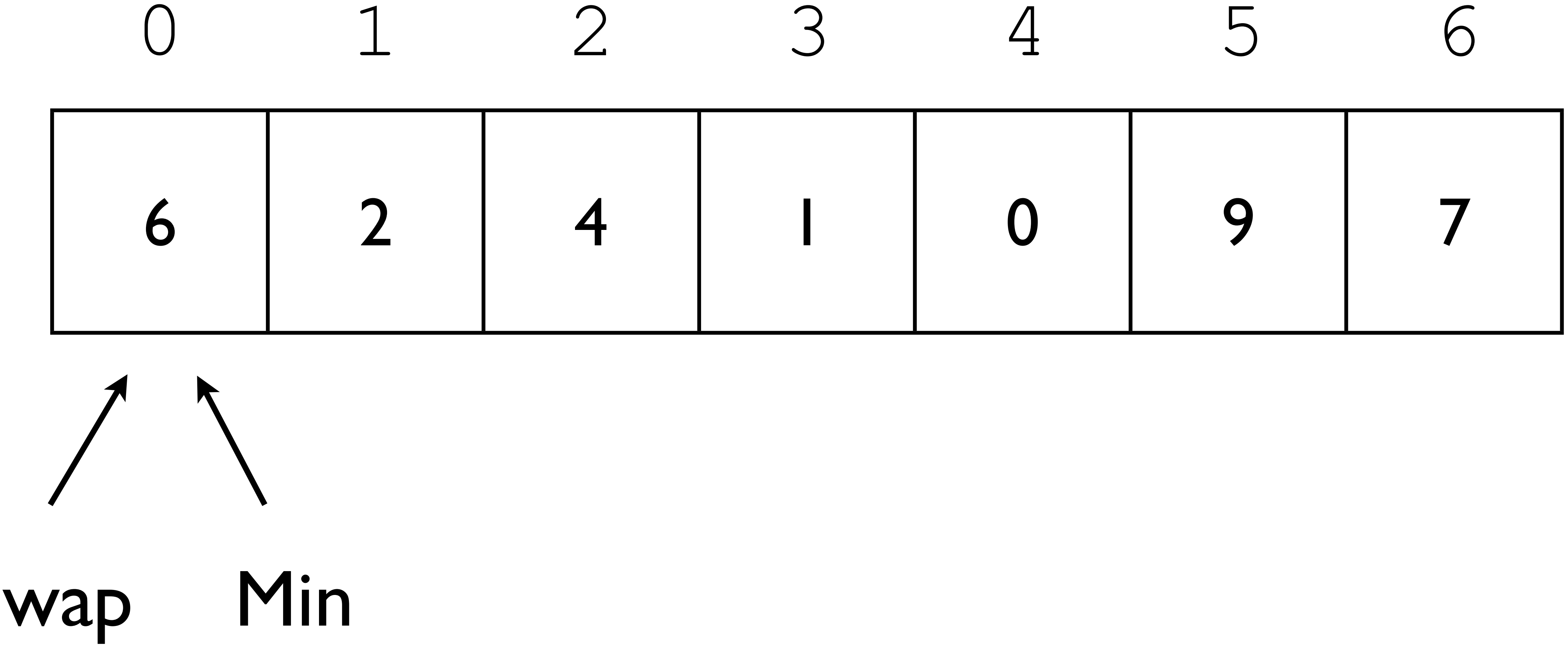
6	2	4	1	0	9	7

Selection Sort

- Basic idea: in an array of length N...
  - Find the minimal element and swap it so that it's in position 0
  - Then, from position 1 and on, find the minimal element and replace it so it's in position 1
  - Keep doing this

Selection Sort

Recorded Minimum Position: **0**
Recorded Minimum: **6**

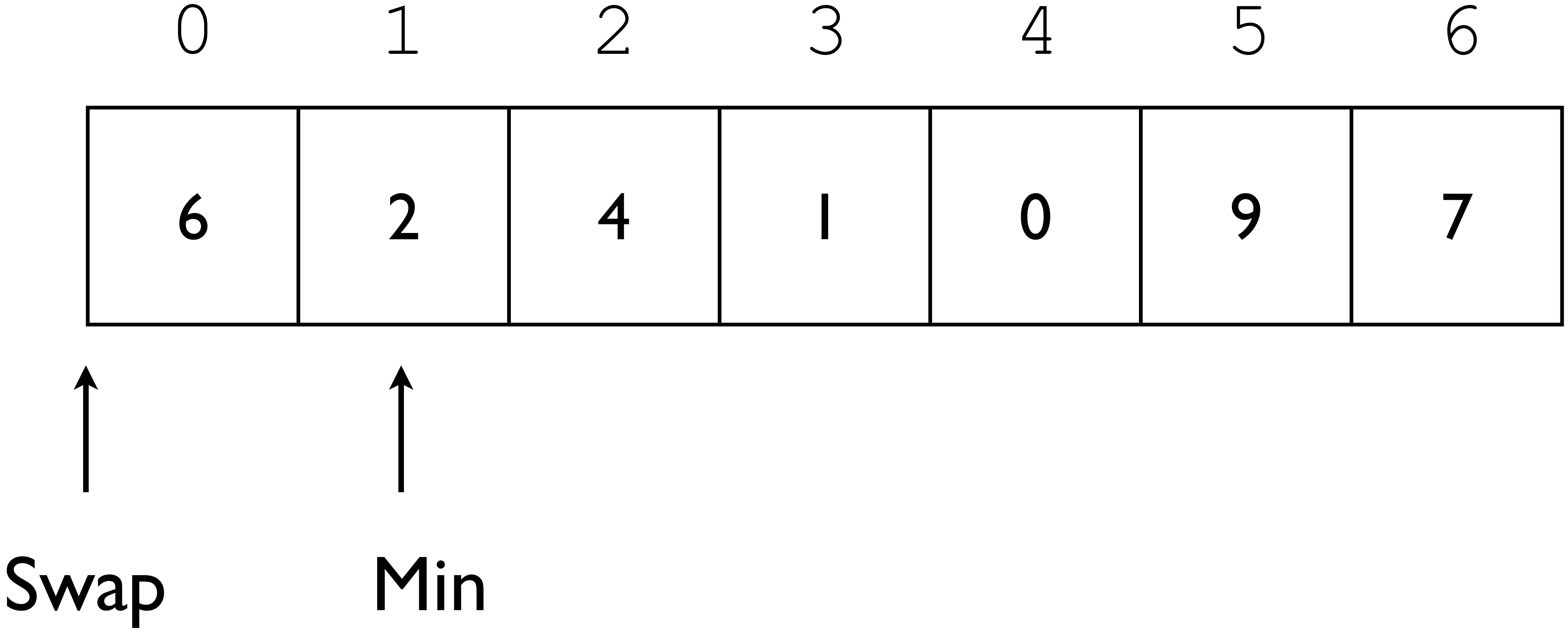


0	1	2	3	4	5	6
6	2	4	1	0	9	7

Selection Sort

Recorded Minimum Position: **1**

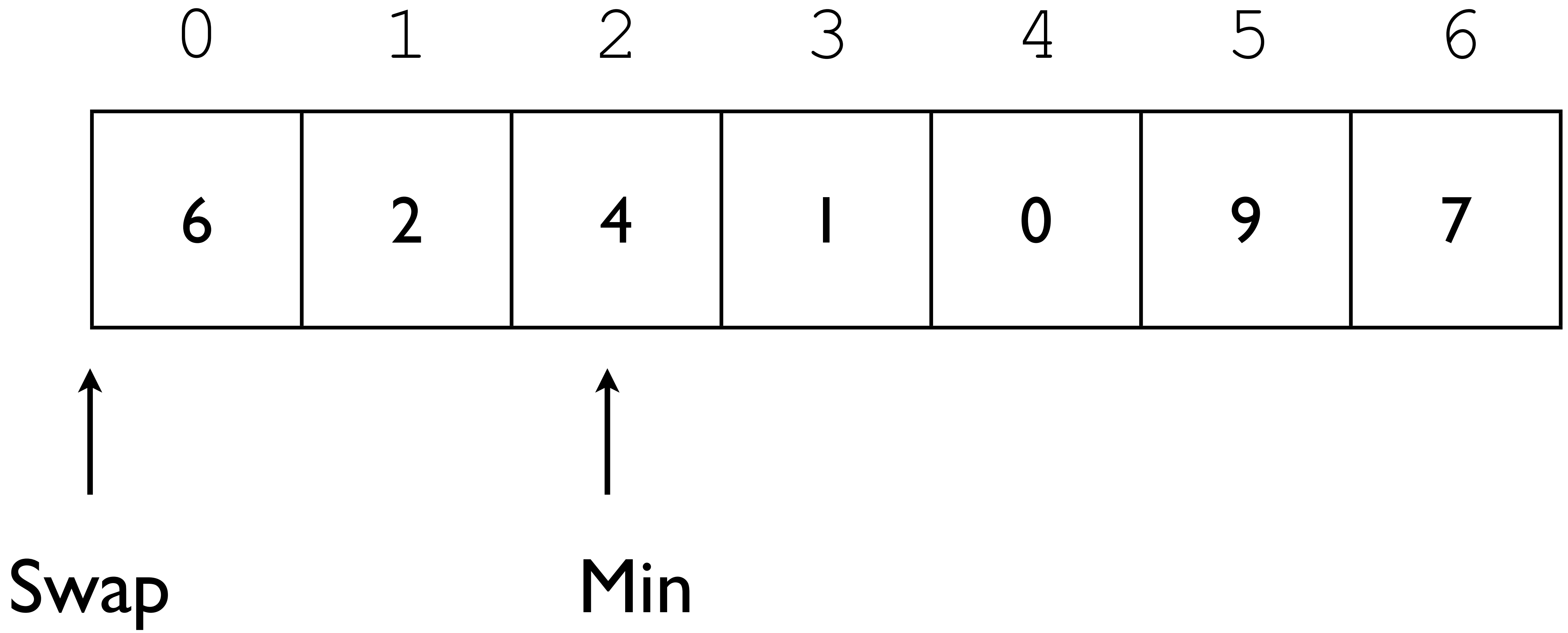
Recorded Minimum: **2**



0	1	2	3	4	5	6
6	2	4	1	0	9	7

Selection Sort

Recorded Minimum Position: 1
Recorded Minimum: 2



0	1	2	3	4	5	6
6	2	4	1	0	9	7

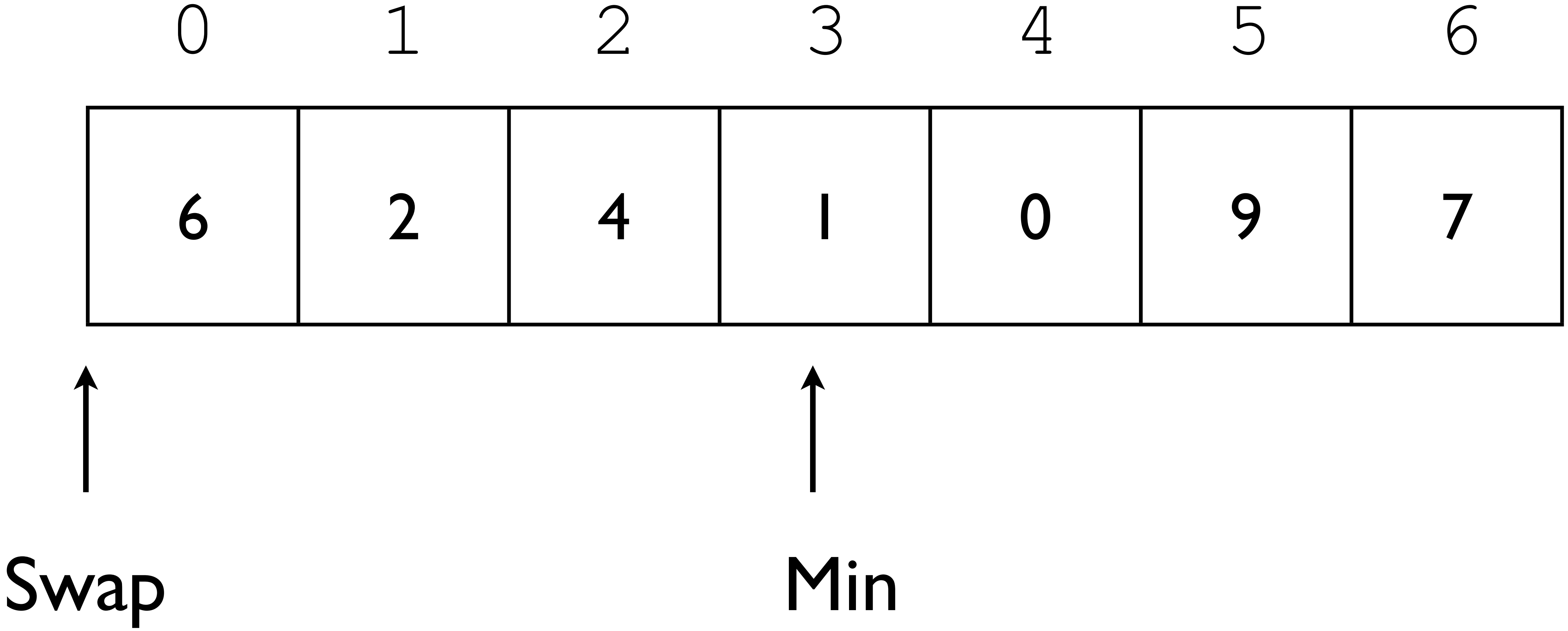
Swap

Min

Selection Sort

Recorded Minimum Position: **3**

Recorded Minimum: **1**



0	1	2	3	4	5	6
6	2	4	1	0	9	7

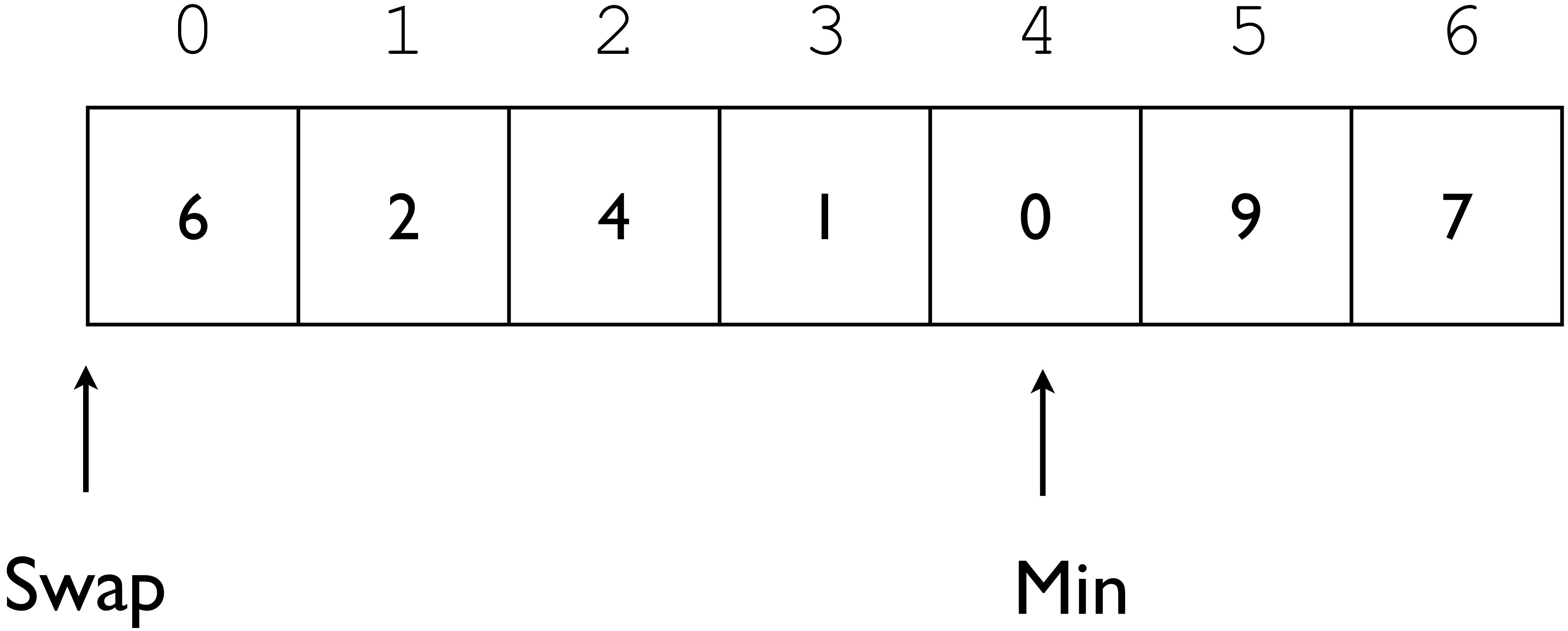
Swap

Min

Selection Sort

Recorded Minimum Position: **4**

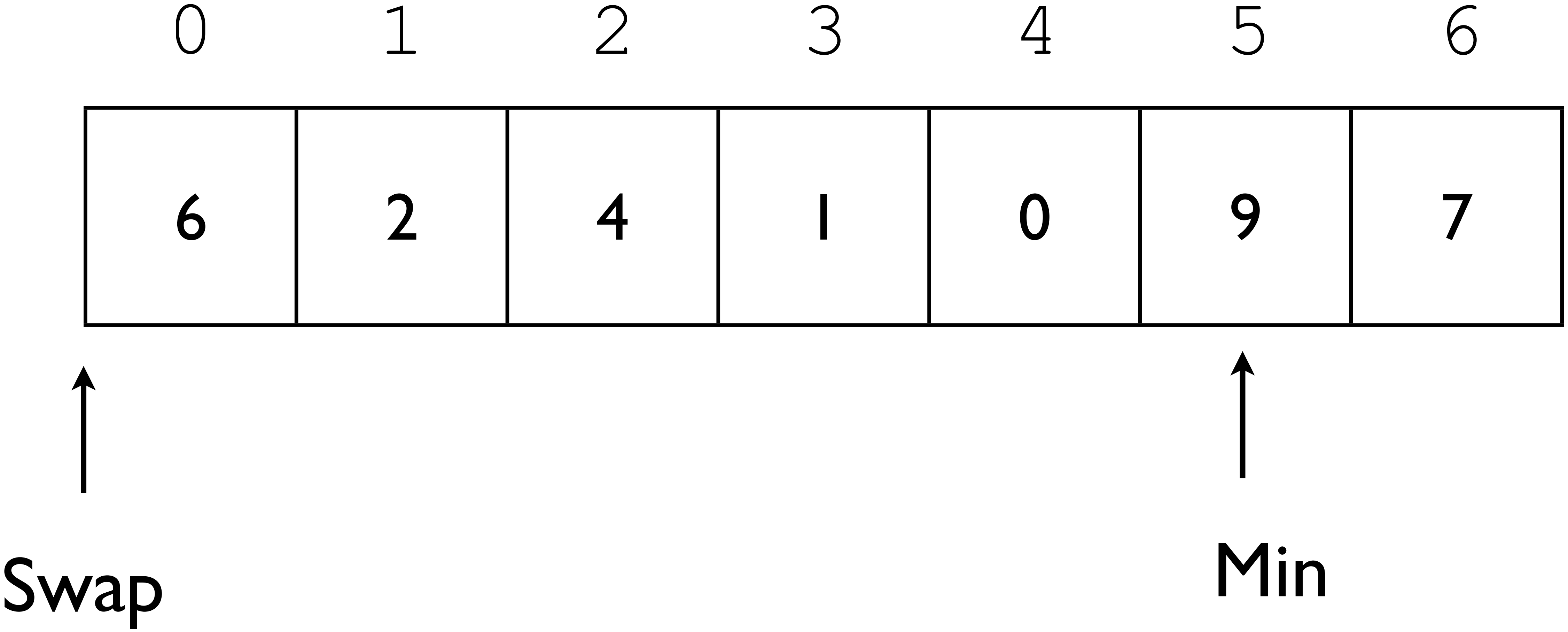
Recorded Minimum: **0**



0	1	2	3	4	5	6
6	2	4	1	0	9	7

Selection Sort

Recorded Minimum Position: 4
Recorded Minimum: 0



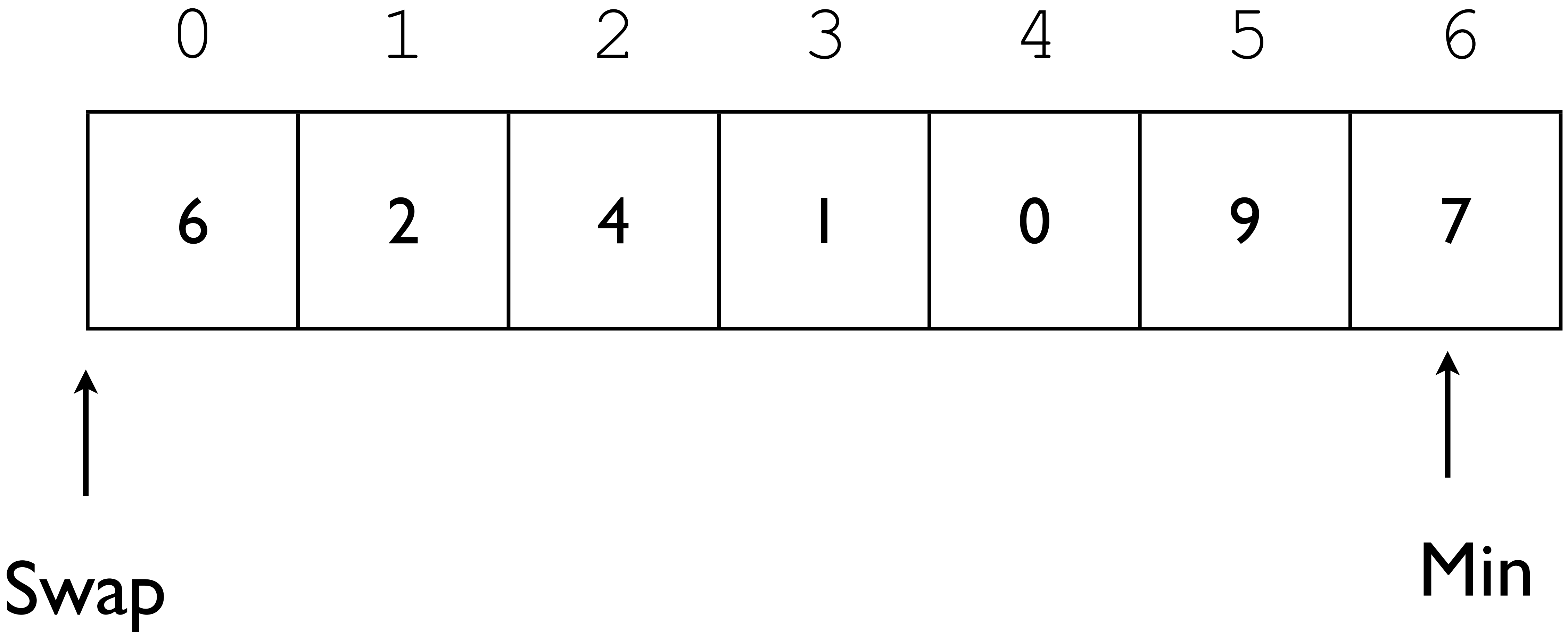
0	1	2	3	4	5	6
6	2	4	1	0	9	7

Swap

Min

Selection Sort

Recorded Minimum Position: 4
Recorded Minimum: 0



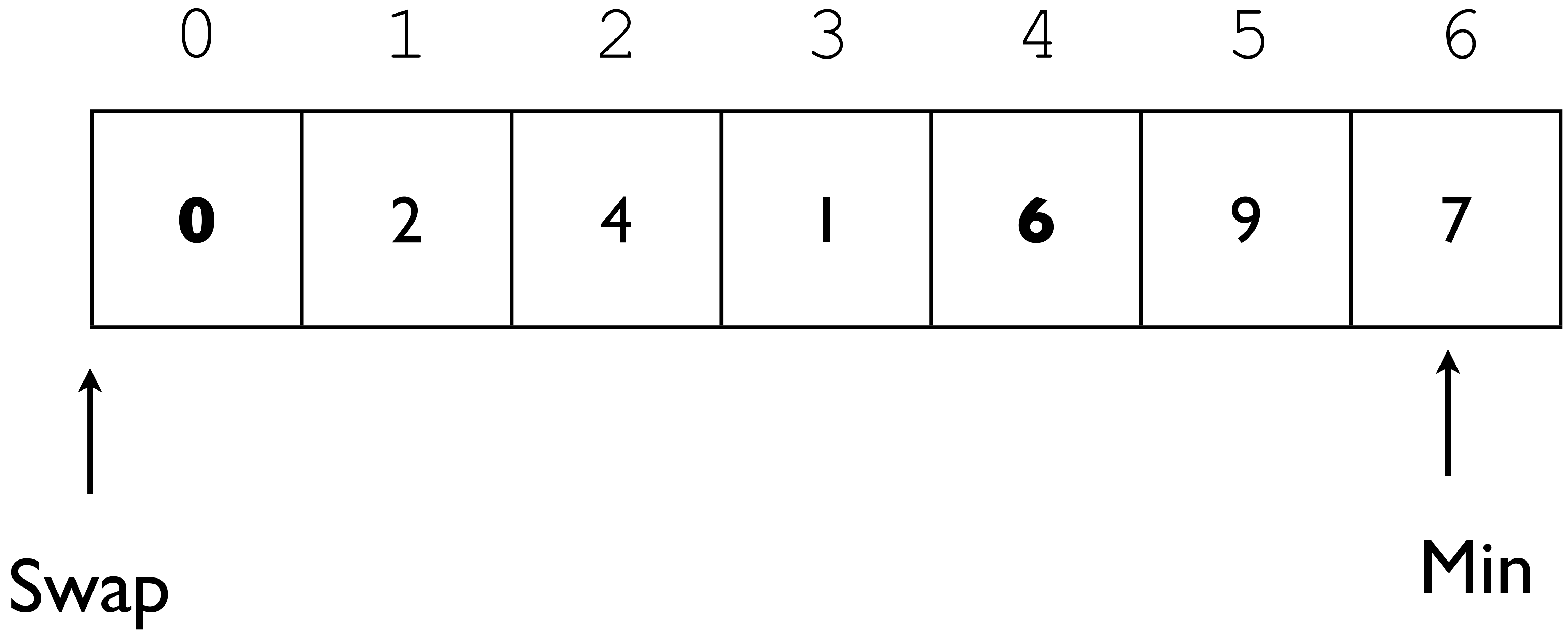
0	1	2	3	4	5	6
6	2	4	1	0	9	7

Swap

Min

Selection Sort

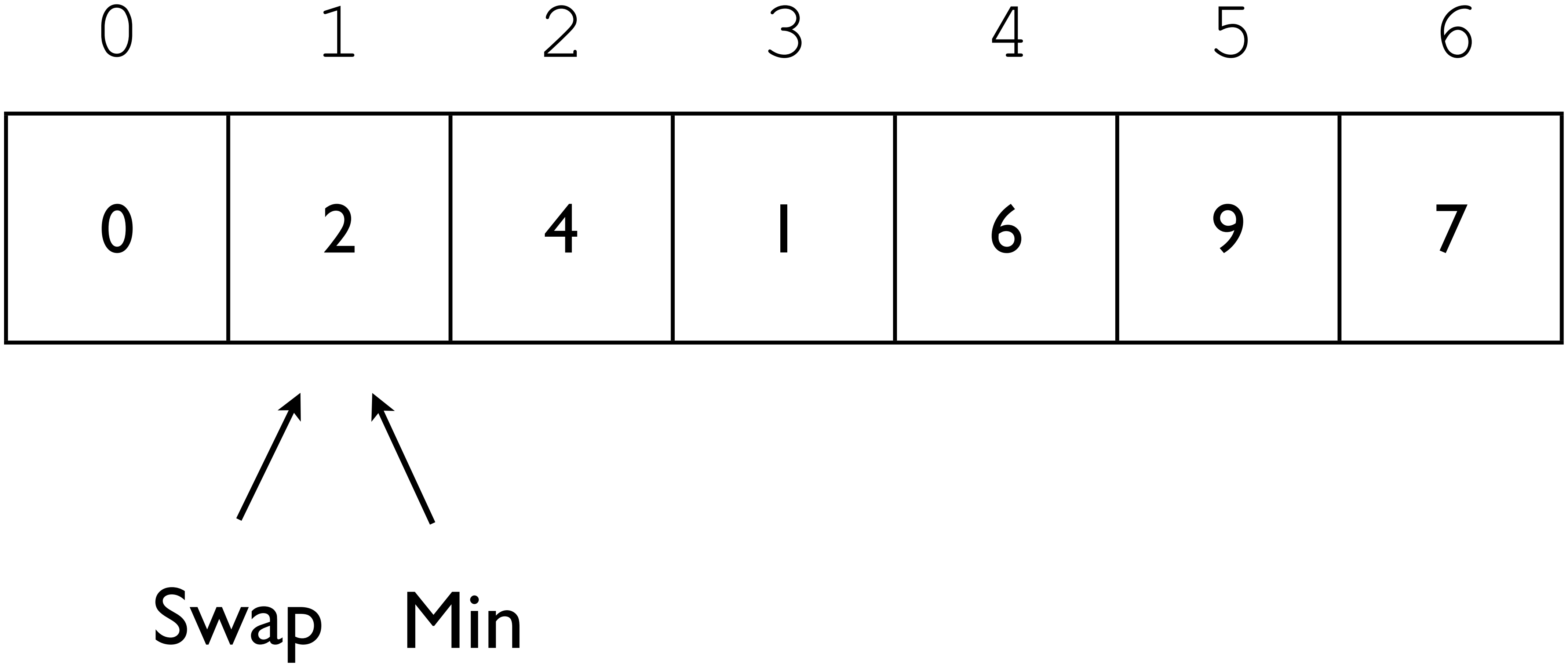
Recorded Minimum Position: 4
Recorded Minimum: 0



0	1	2	3	4	5	6
0	2	4	1	**6**	9	7

Selection Sort

Recorded Minimum Position: **1**
Recorded Minimum: **2**

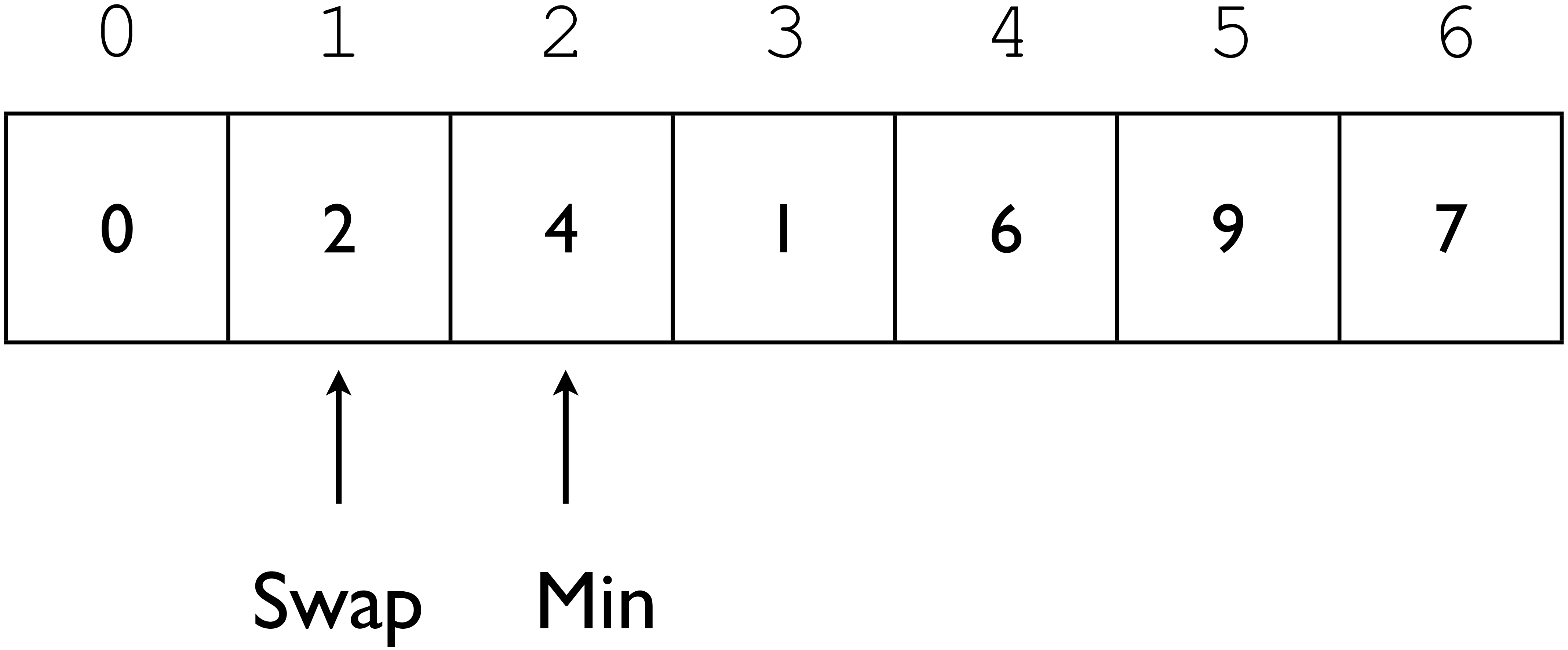


0	1	2	3	4	5	6
0	2	4	1	6	9	7

Swap Min

Selection Sort

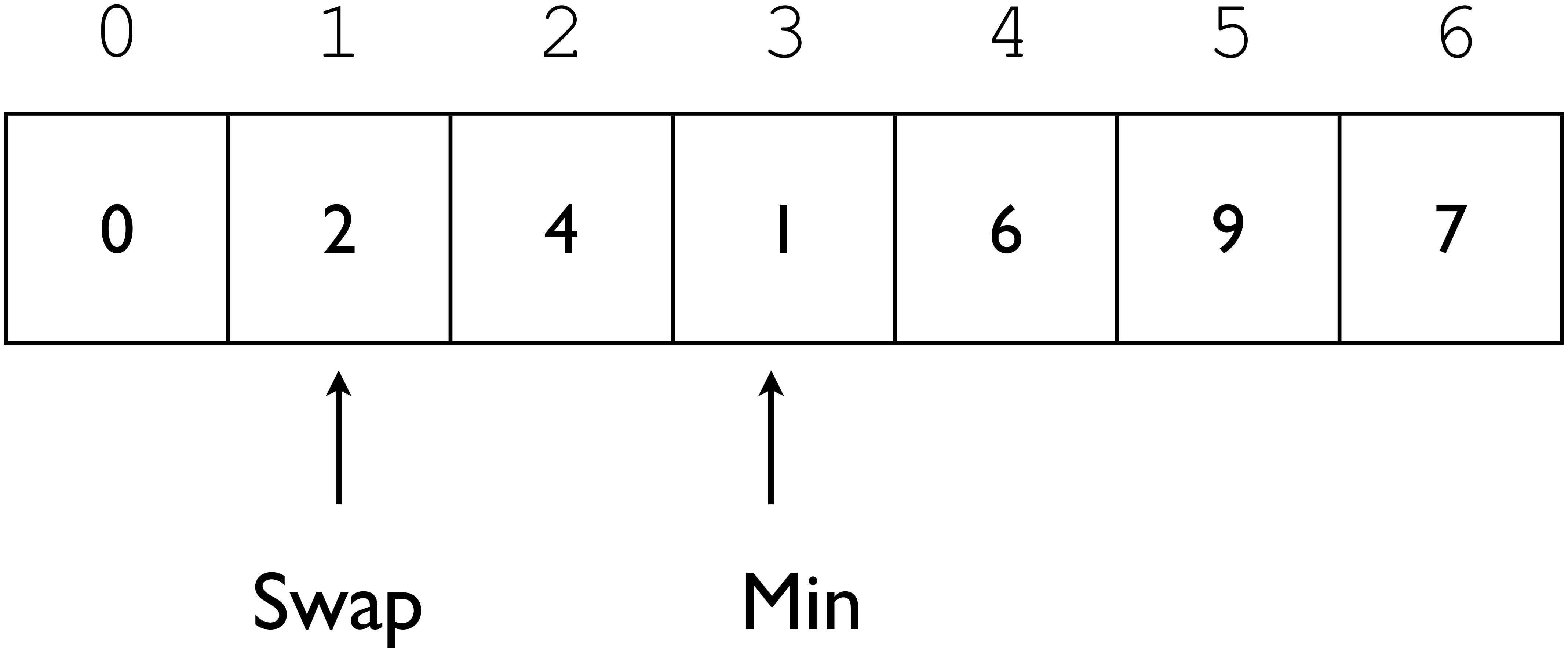
Recorded Minimum Position: 1
Recorded Minimum: 2



0	1	2	3	4	5	6
0	2	4	1	6	9	7

Selection Sort

Recorded Minimum Position: **3**
Recorded Minimum: **1**



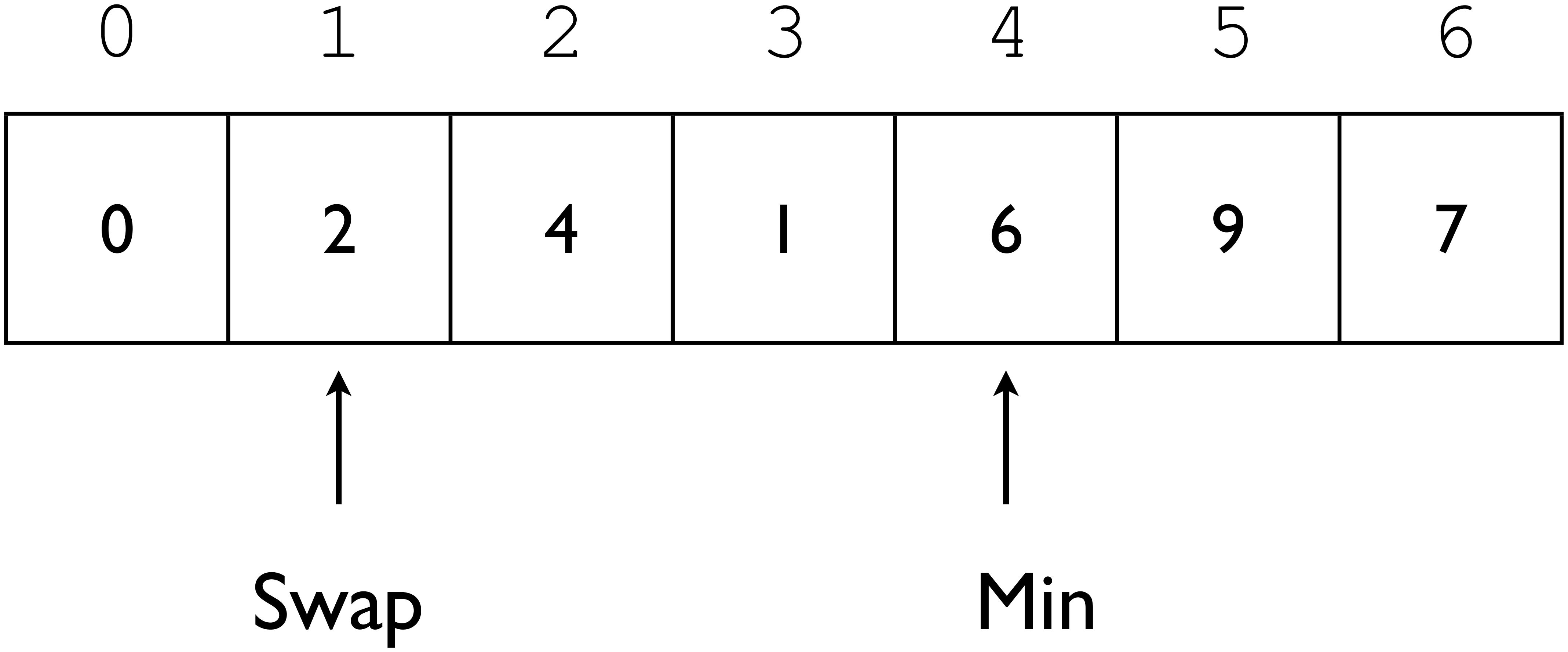
0	1	2	3	4	5	6
0	2	4	1	6	9	7

Swap

Min

Selection Sort

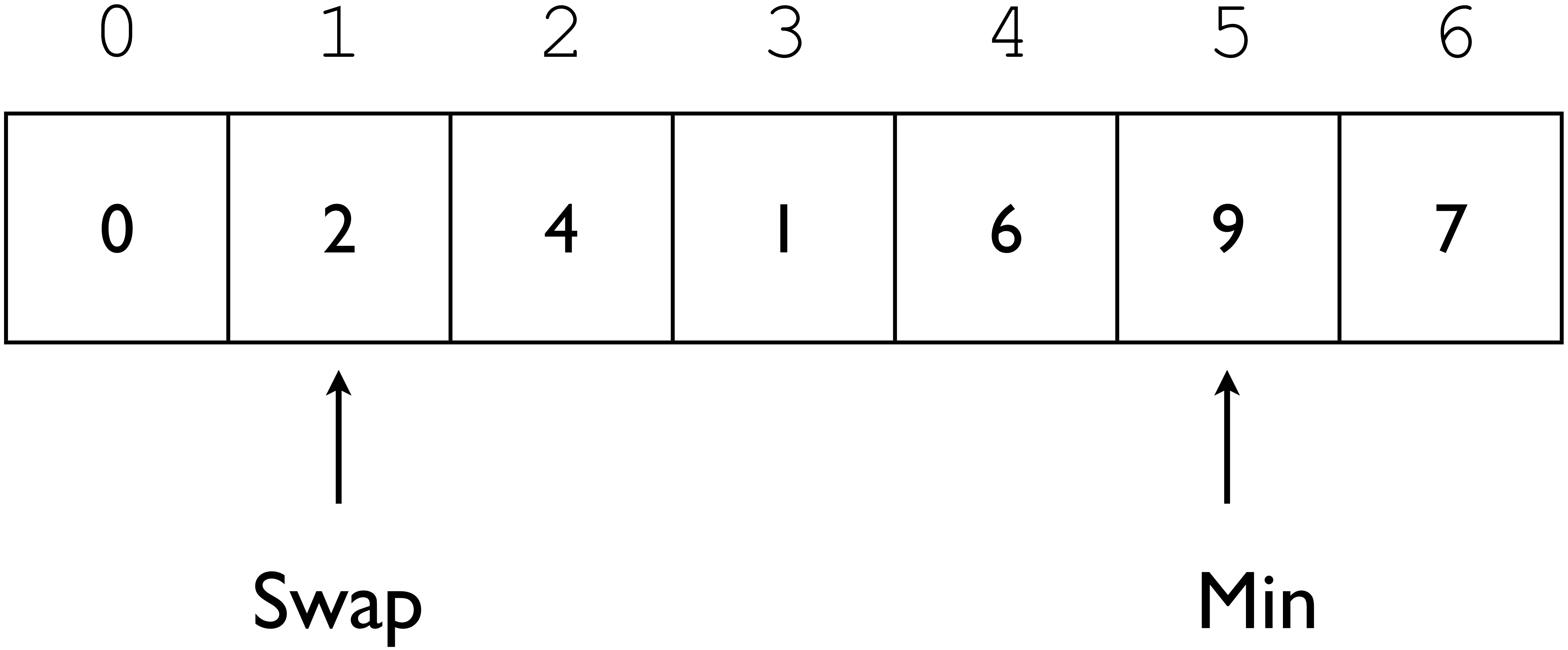
Recorded Minimum Position: 3
Recorded Minimum: 1



0	1	2	3	4	5	6
0	2	4	1	6	9	7

Selection Sort

Recorded Minimum Position: 3
Recorded Minimum: 1



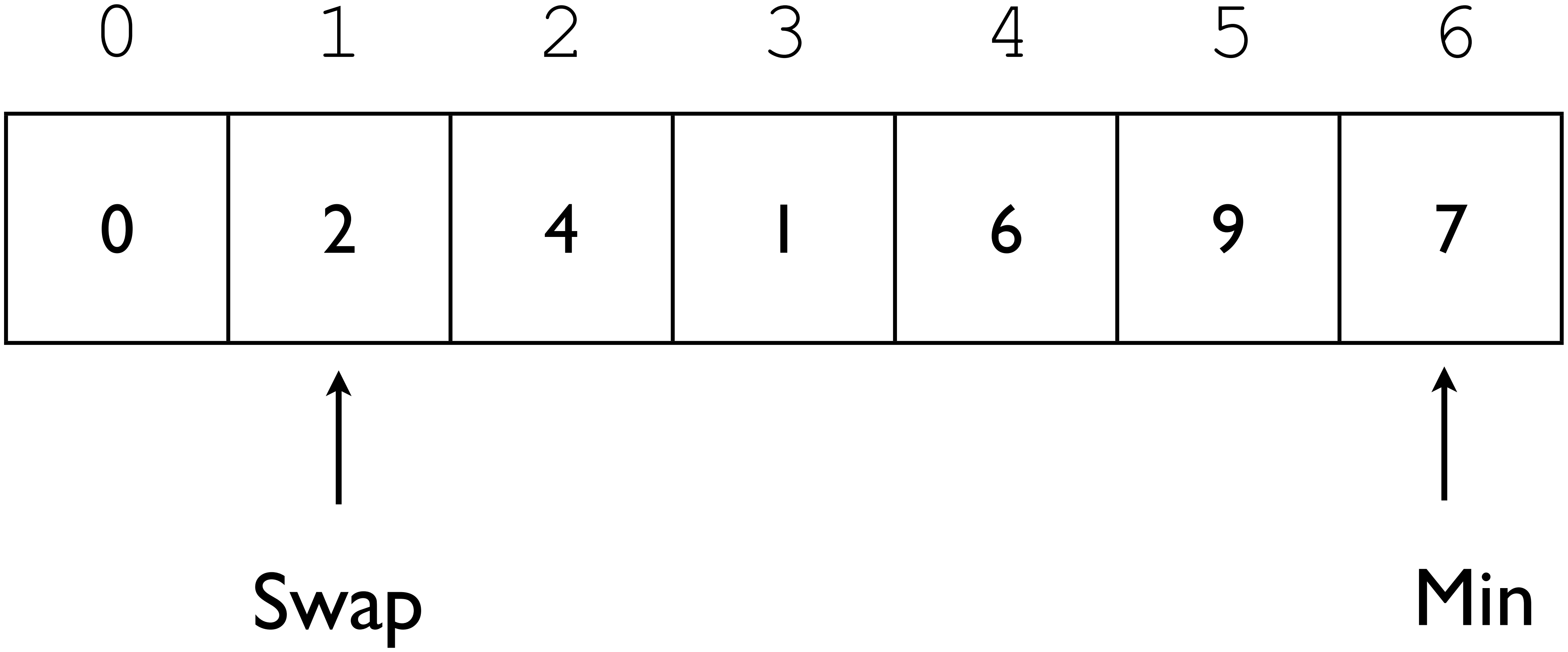
0	1	2	3	4	5	6
0	2	4	1	6	9	7

Swap

Min

Selection Sort

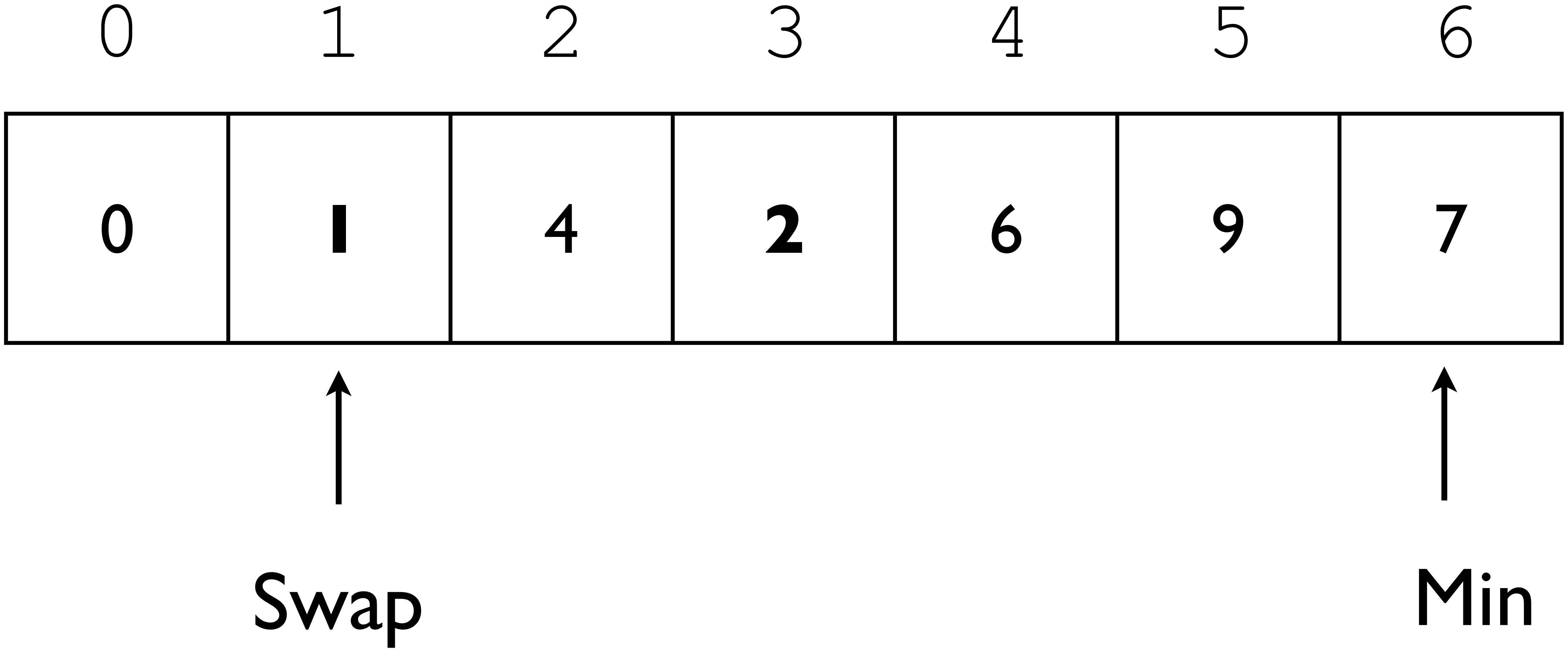
Recorded Minimum Position: 3
Recorded Minimum: 1



0	1	2	3	4	5	6
0	2	4	1	6	9	7

Selection Sort

Recorded Minimum Position: 3
Recorded Minimum: 1

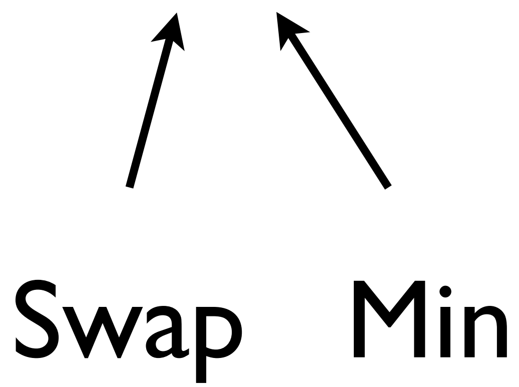


0	1	2	3	4	5	6
0	**1**	**4**	**2**	**6**	**9**	**7**

Selection Sort

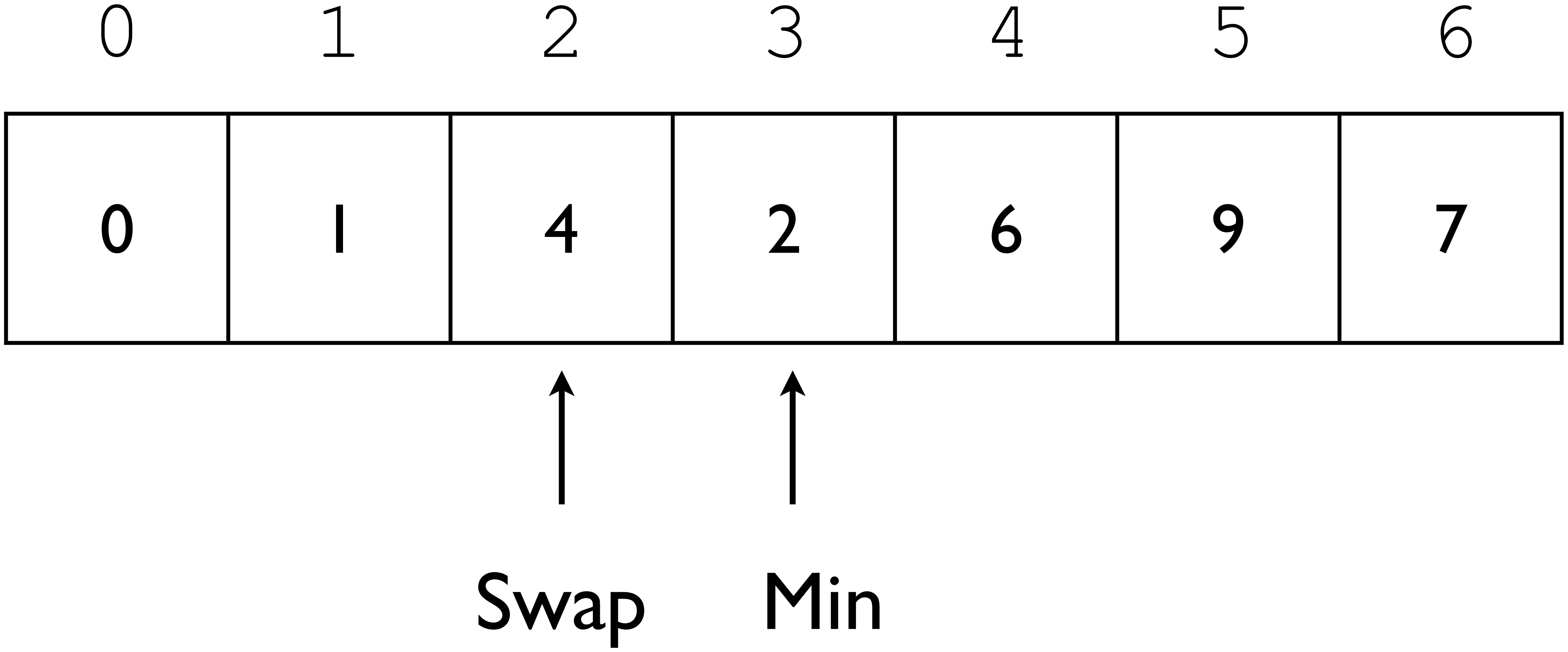
Recorded Minimum Position: **2**
Recorded Minimum: **4**

0	1	2	3	4	5	6
0	1	4	2	6	9	7



Selection Sort

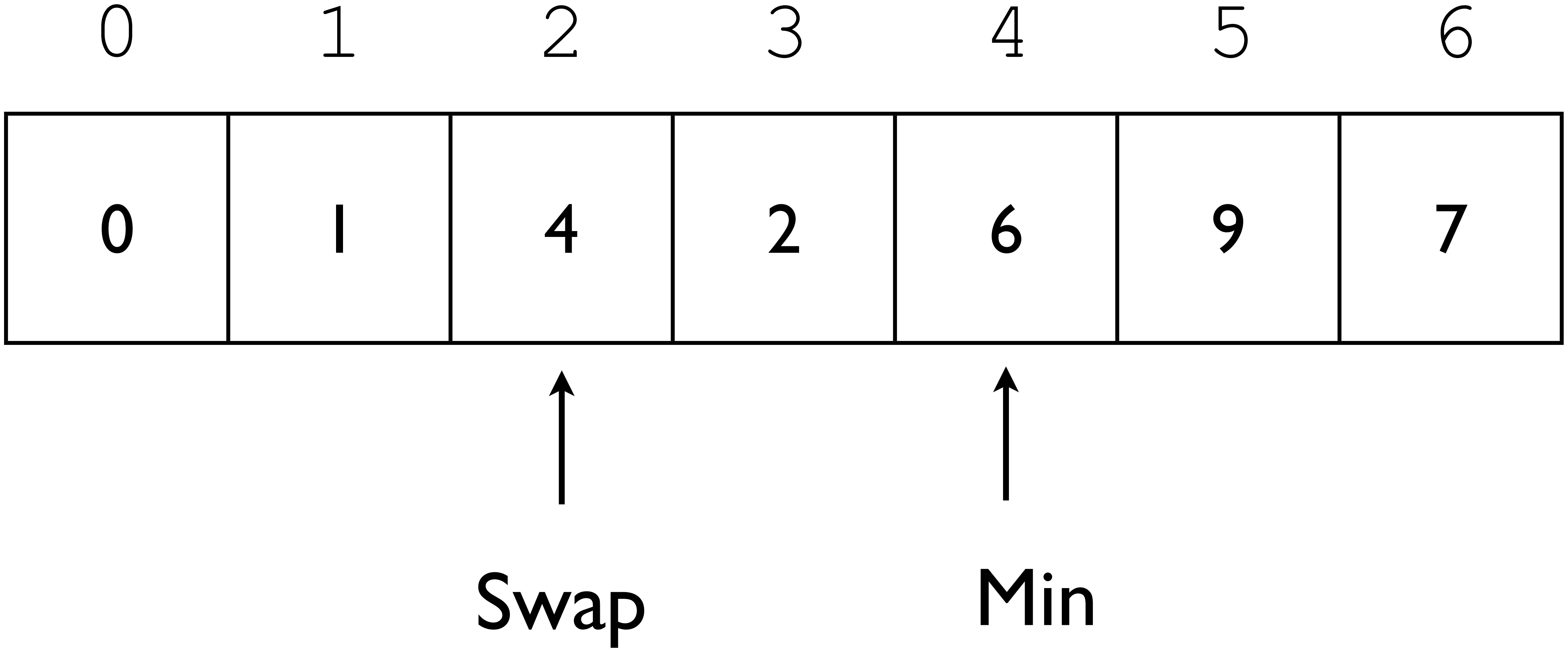
Recorded Minimum Position: **3**
Recorded Minimum: **3**



0	1	2	3	4	5	6
0	1	4	2	6	9	7

Selection Sort

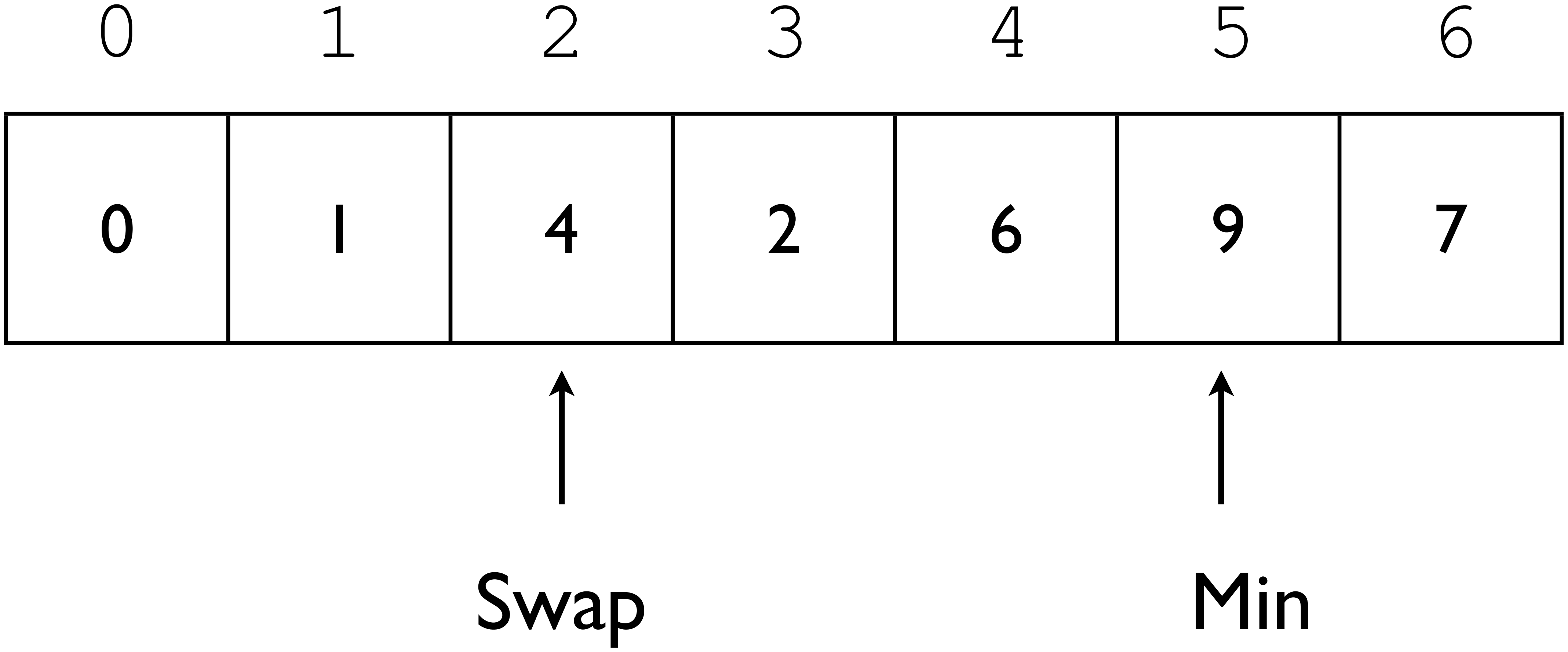
Recorded Minimum Position: 3
Recorded Minimum: 3



0	1	2	3	4	5	6
0	1	4	2	6	9	7

Selection Sort

Recorded Minimum Position: 3
Recorded Minimum: 3



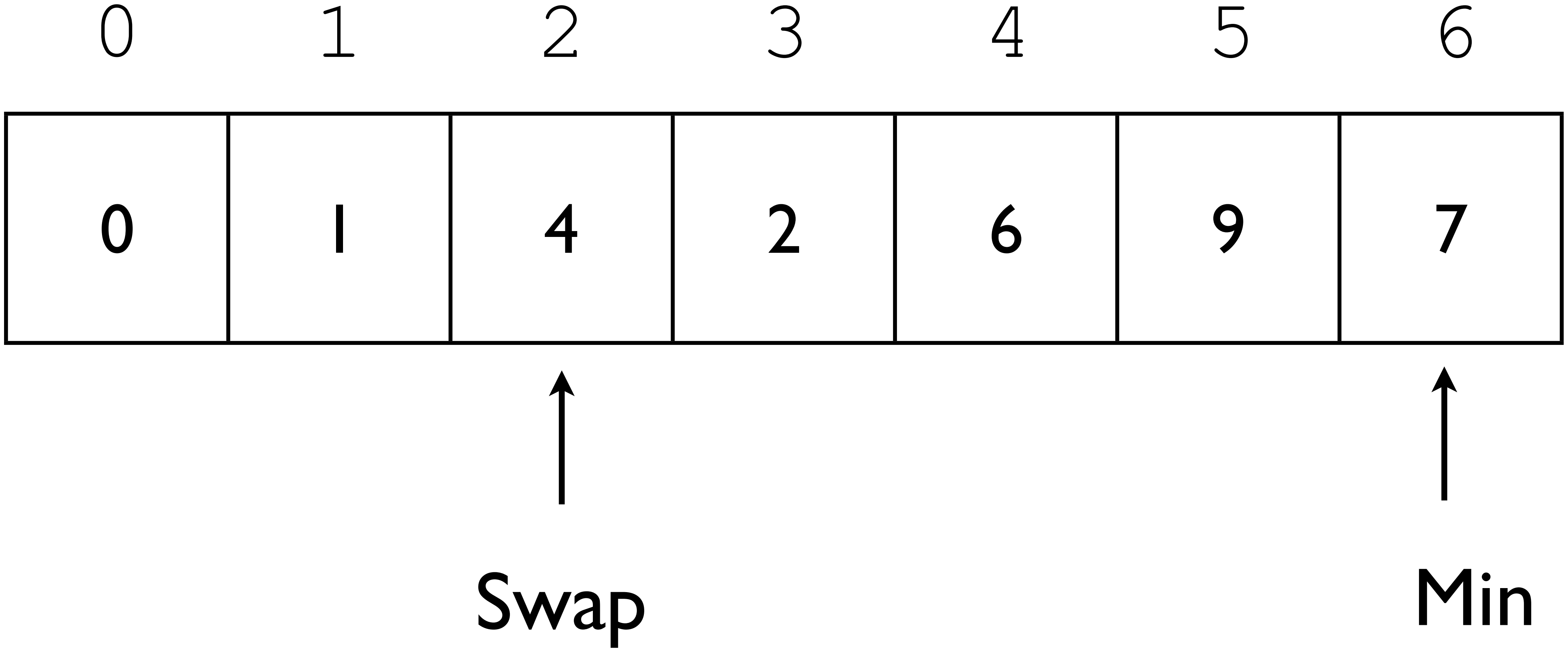
0	1	2	3	4	5	6
0	1	4	2	6	9	7

Swap

Min

Selection Sort

Recorded Minimum Position: 3
Recorded Minimum: 3



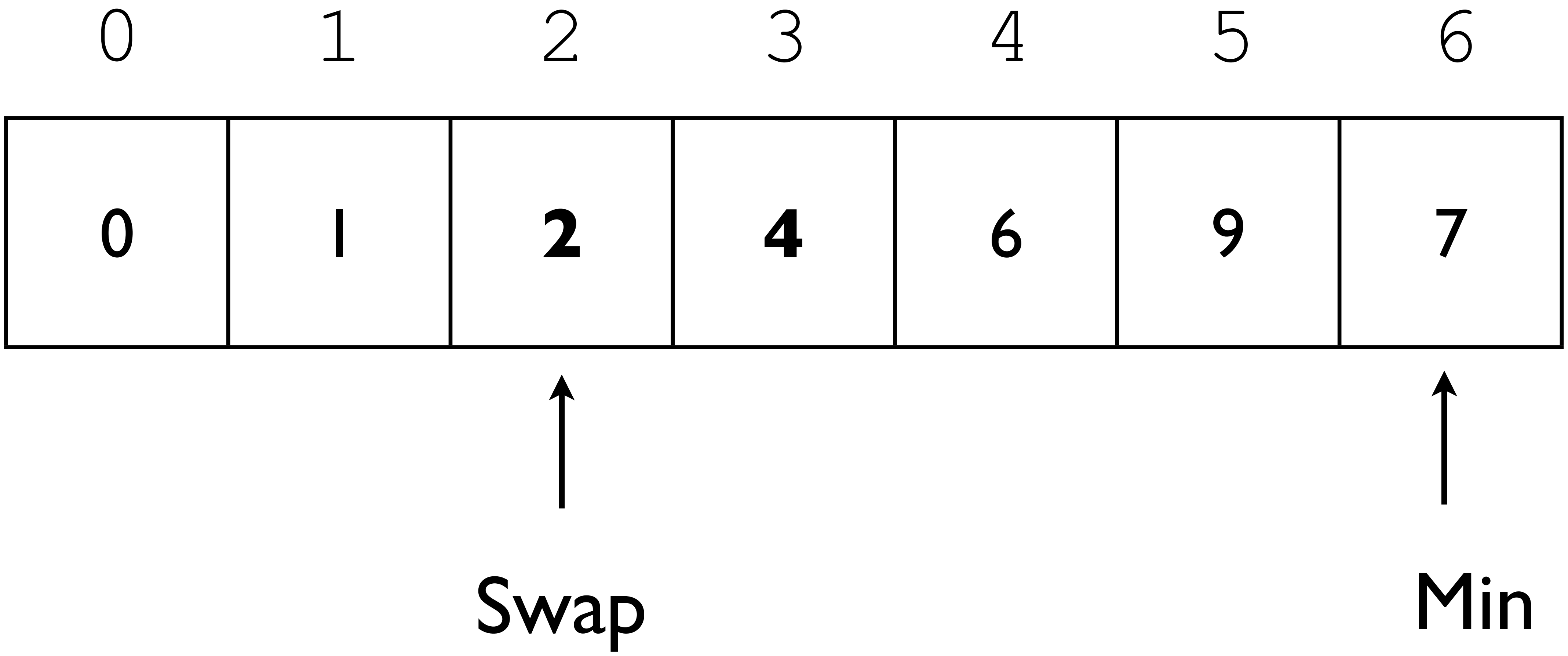
0	1	2	3	4	5	6
0	1	4	2	6	9	7

Swap

Min

Selection Sort

Recorded Minimum Position: 3
Recorded Minimum: 3



0	1	2	3	4	5	6
0	1	**2**	**4**	6	9	7

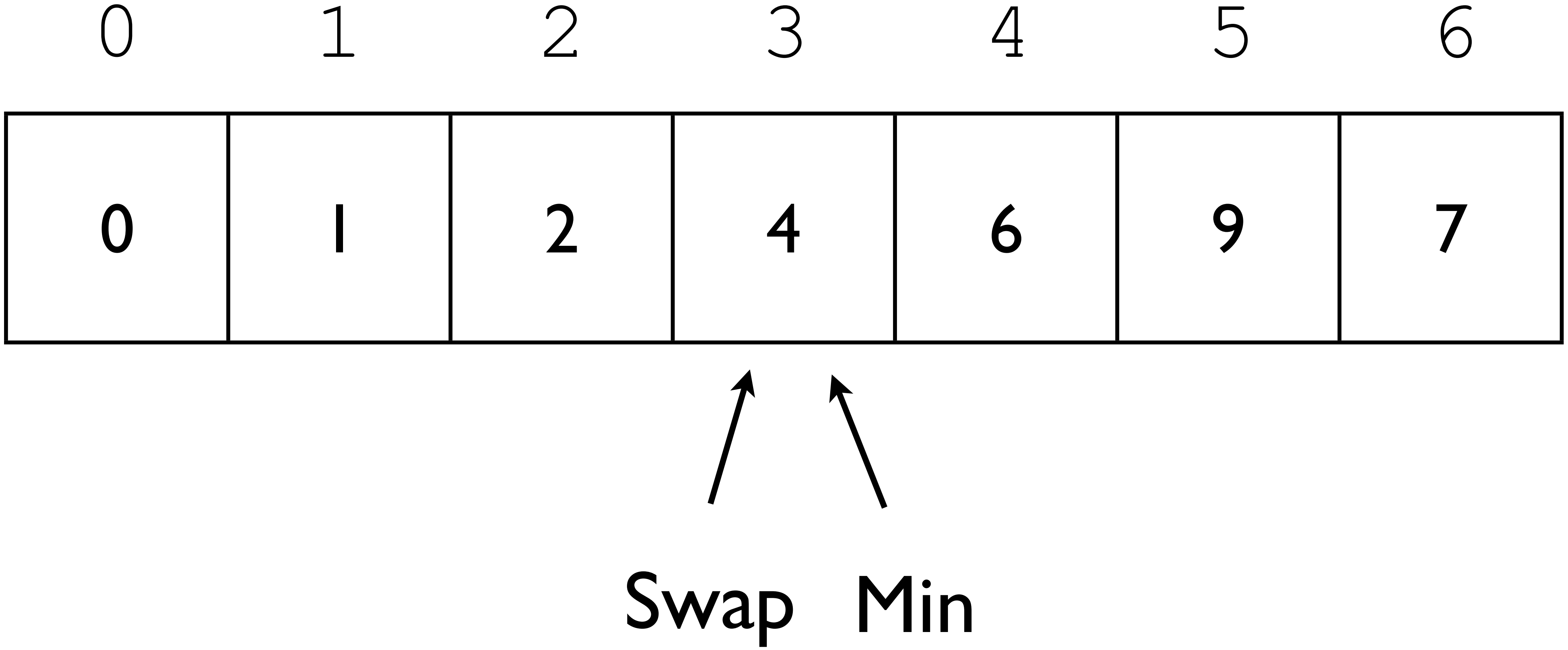
Swap

Min

Selection Sort

Recorded Minimum Position: **3**

Recorded Minimum: **4**

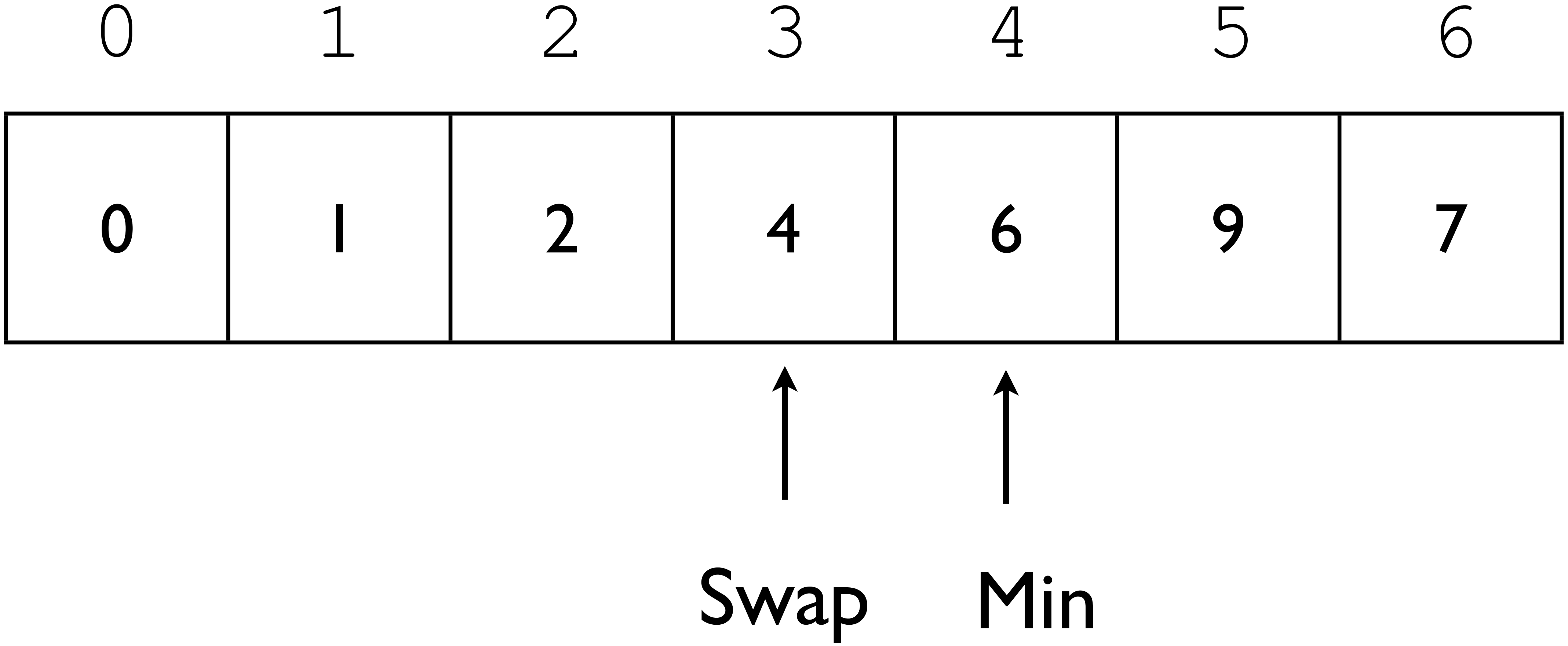


0	1	2	3	4	5	6
0	1	2	4	6	9	7

Swap Min

Selection Sort

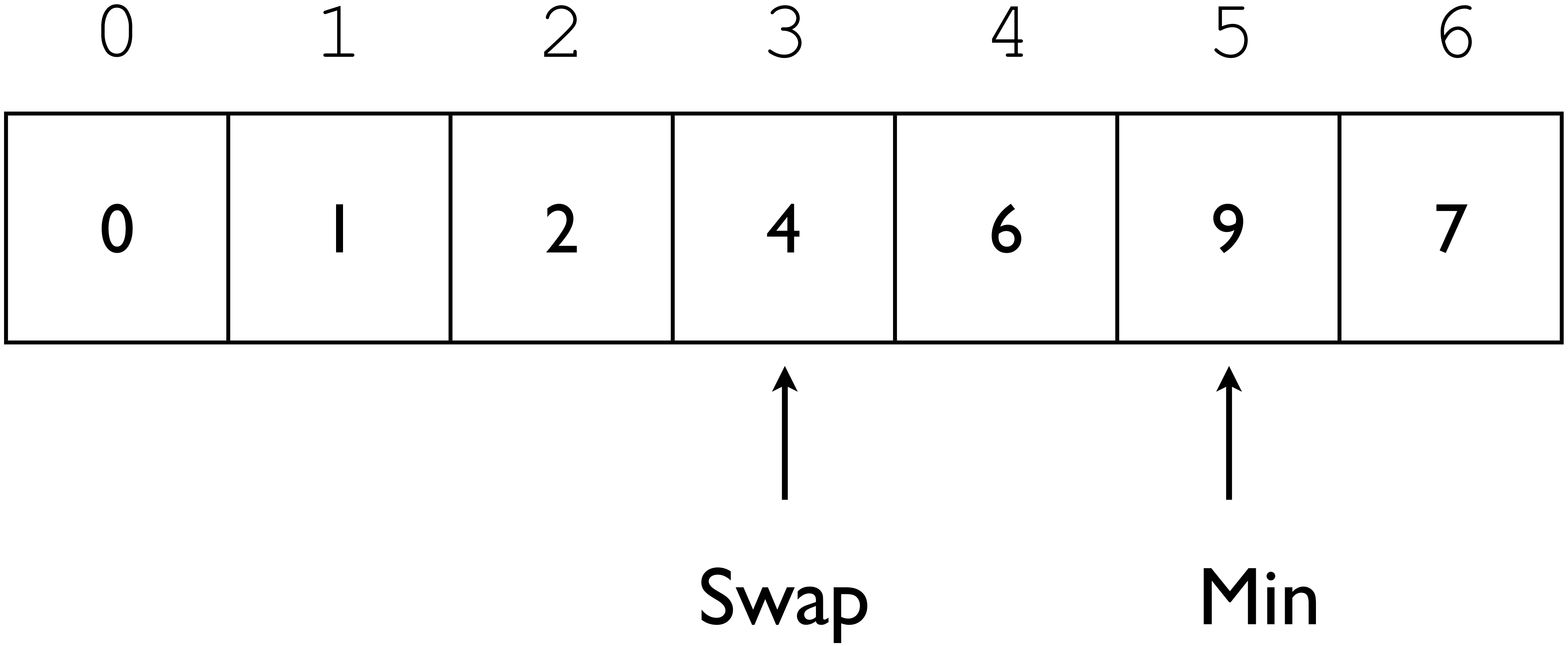
Recorded Minimum Position: 3
Recorded Minimum: 4



0	1	2	3	4	5	6
0	1	2	4	6	9	7

Selection Sort

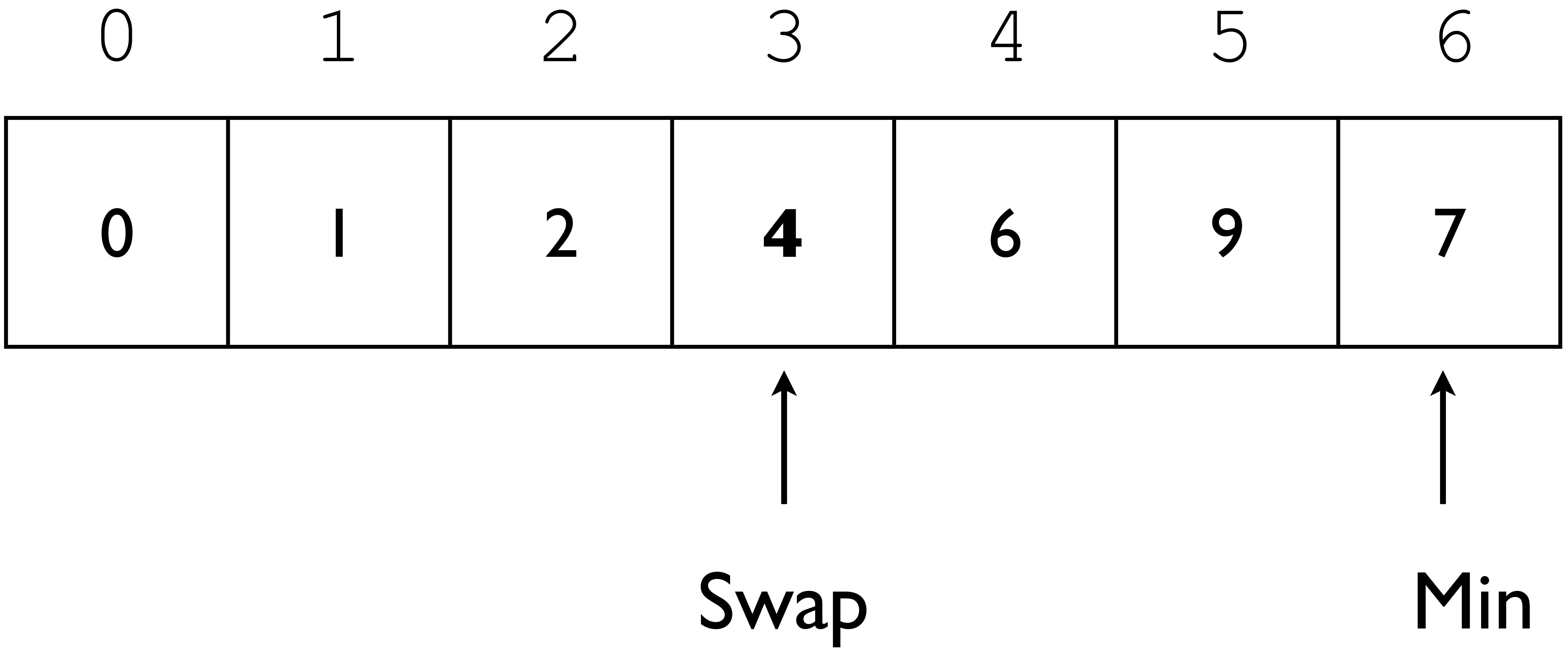
Recorded Minimum Position: 3
Recorded Minimum: 4



0	1	2	3	4	5	6
0	1	2	4	6	9	7

Selection Sort

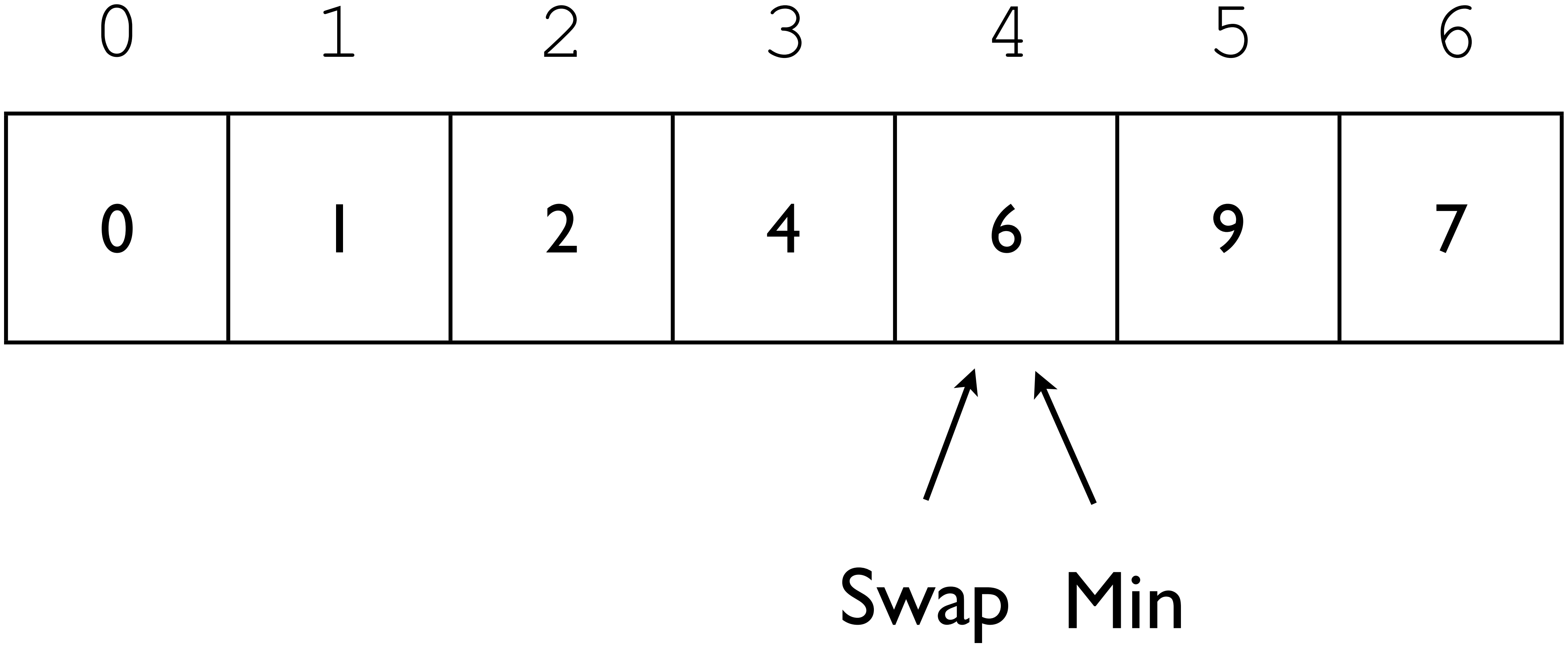
Recorded Minimum Position: 3
Recorded Minimum: 4



0	1	2	3	4	5	6
0	1	2	4	6	9	7

Selection Sort

Recorded Minimum Position: **4**
Recorded Minimum: **6**

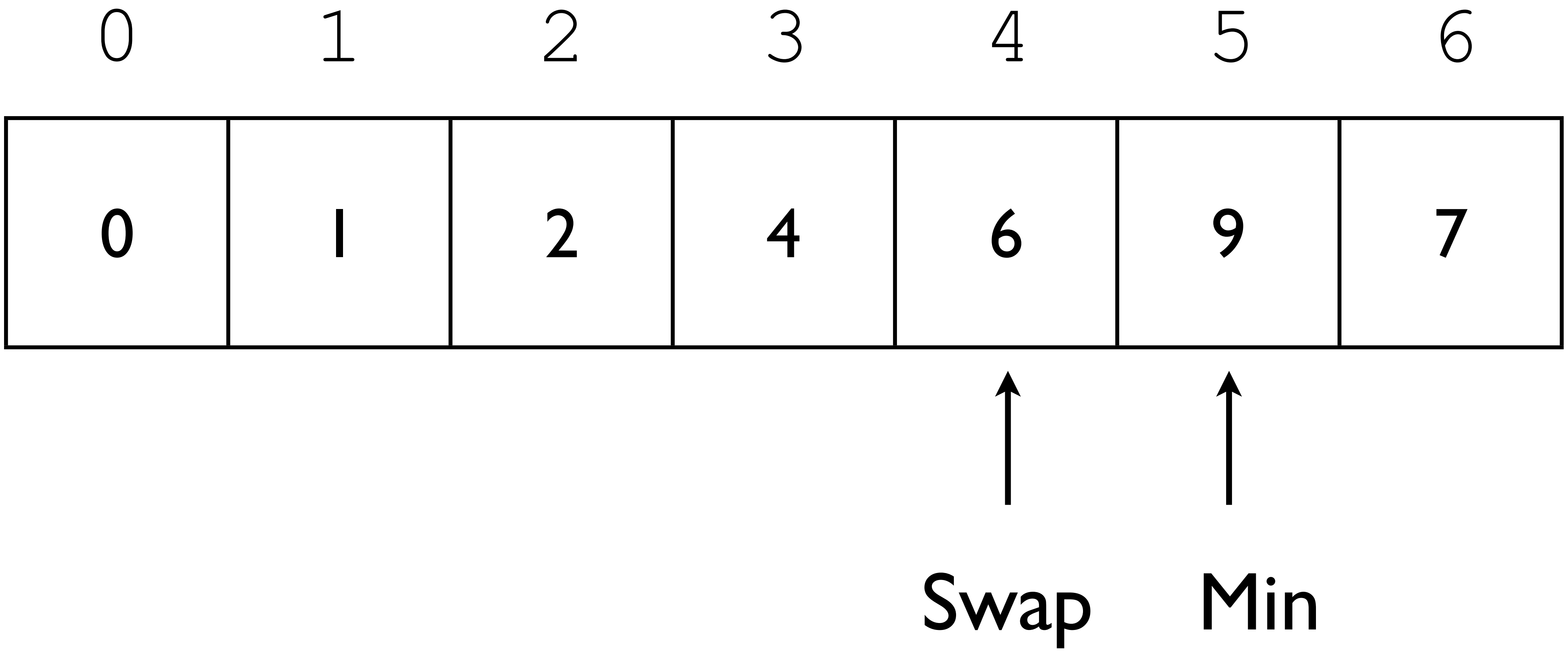


0	1	2	3	4	5	6
0	1	2	4	6	9	7

Swap Min

Selection Sort

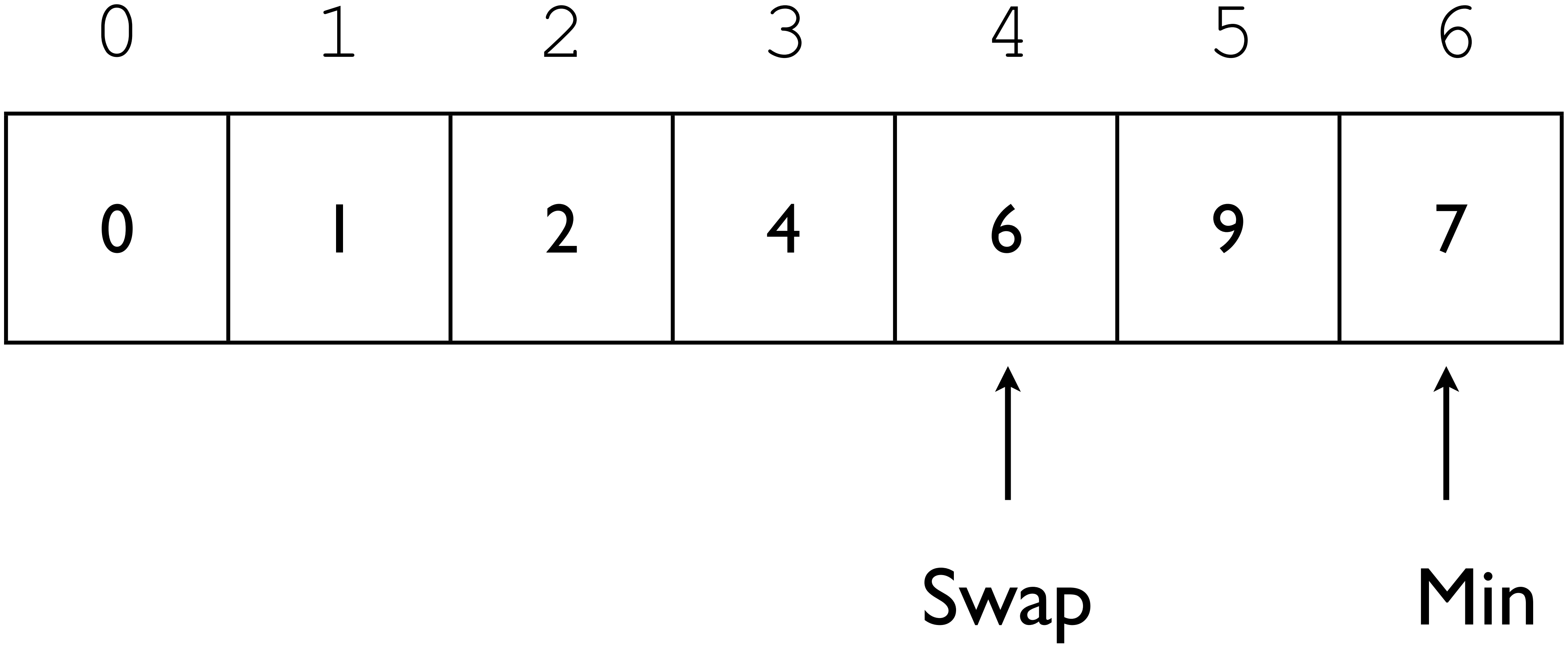
Recorded Minimum Position: 4
Recorded Minimum: 6



0	1	2	3	4	5	6
0	1	2	4	6	9	7

Selection Sort

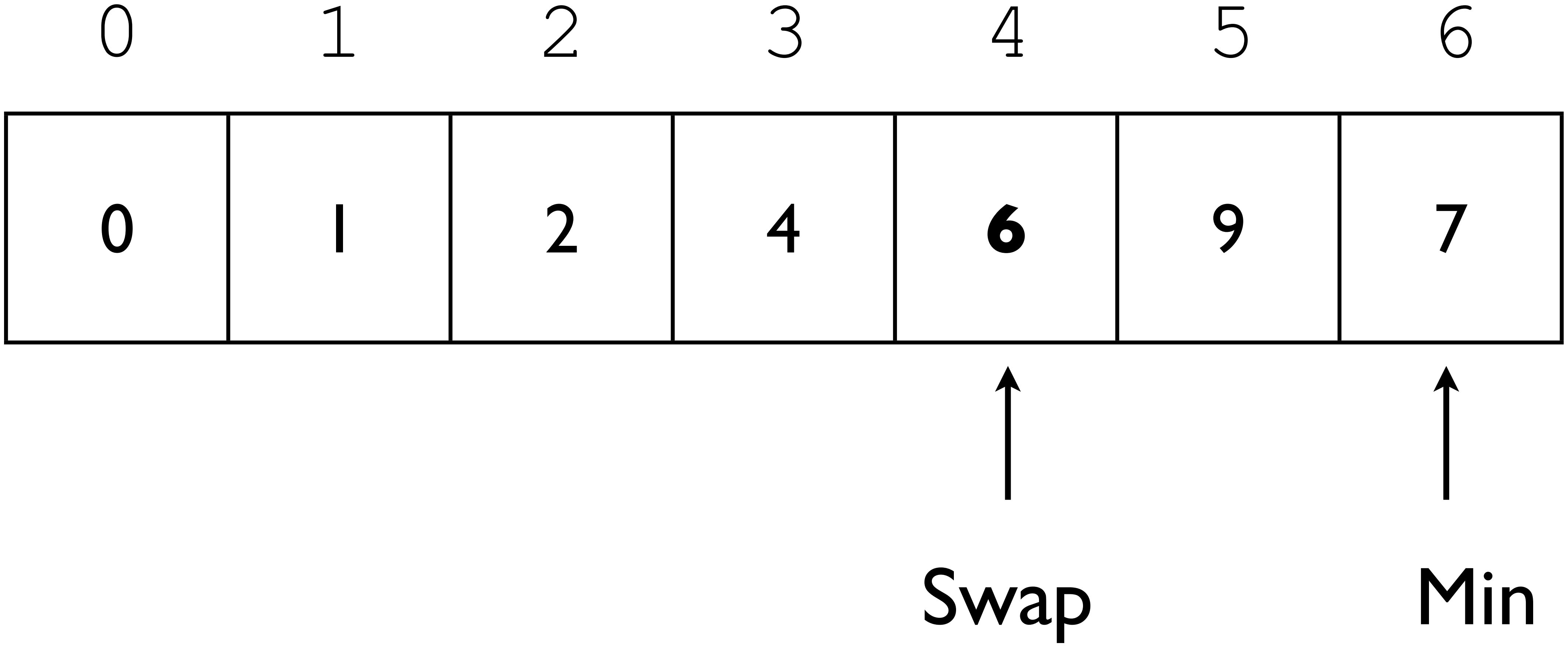
Recorded Minimum Position: 4
Recorded Minimum: 6



0	1	2	3	4	5	6
0	1	2	4	6	9	7

Selection Sort

Recorded Minimum Position: 4
Recorded Minimum: 6

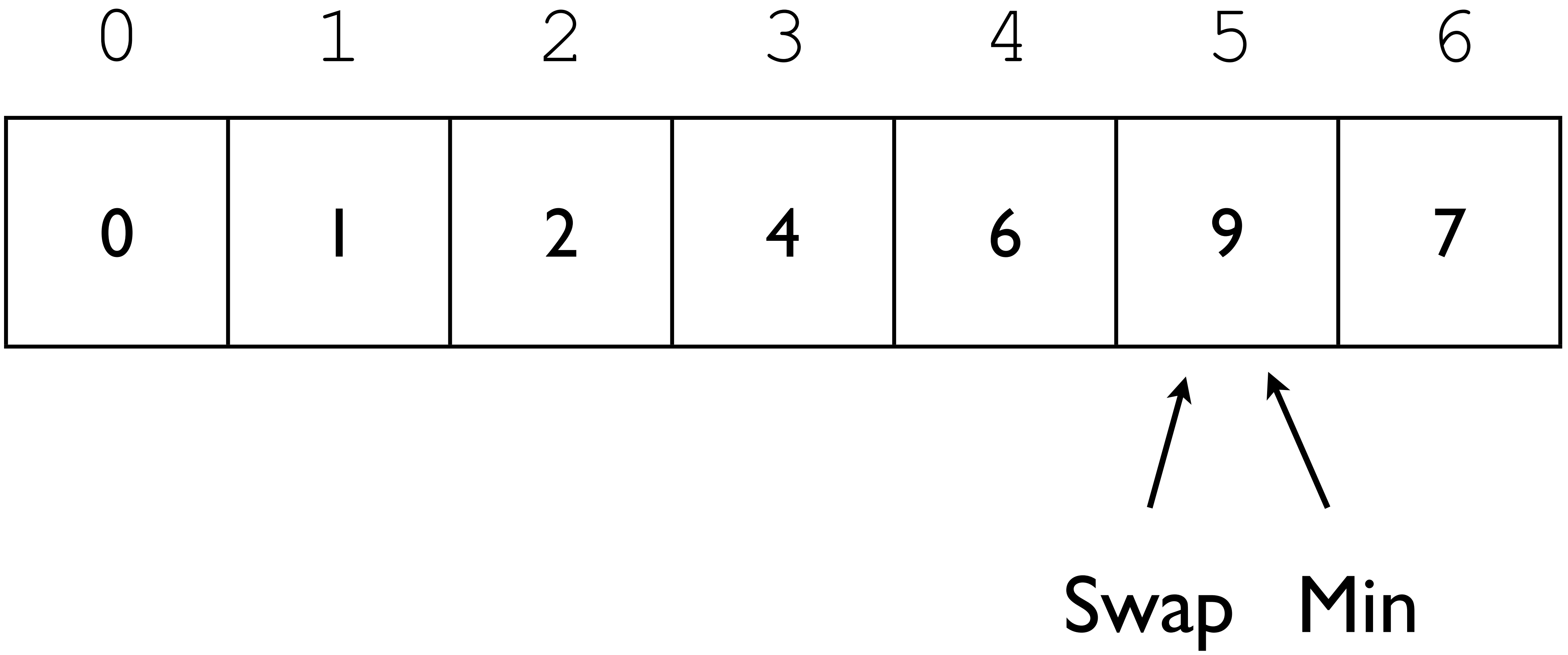


0	1	2	3	4	5	6
0	**1**	**2**	**4**	**6**	**9**	**7**

Selection Sort

Recorded Minimum Position: **5**

Recorded Minimum: **9**

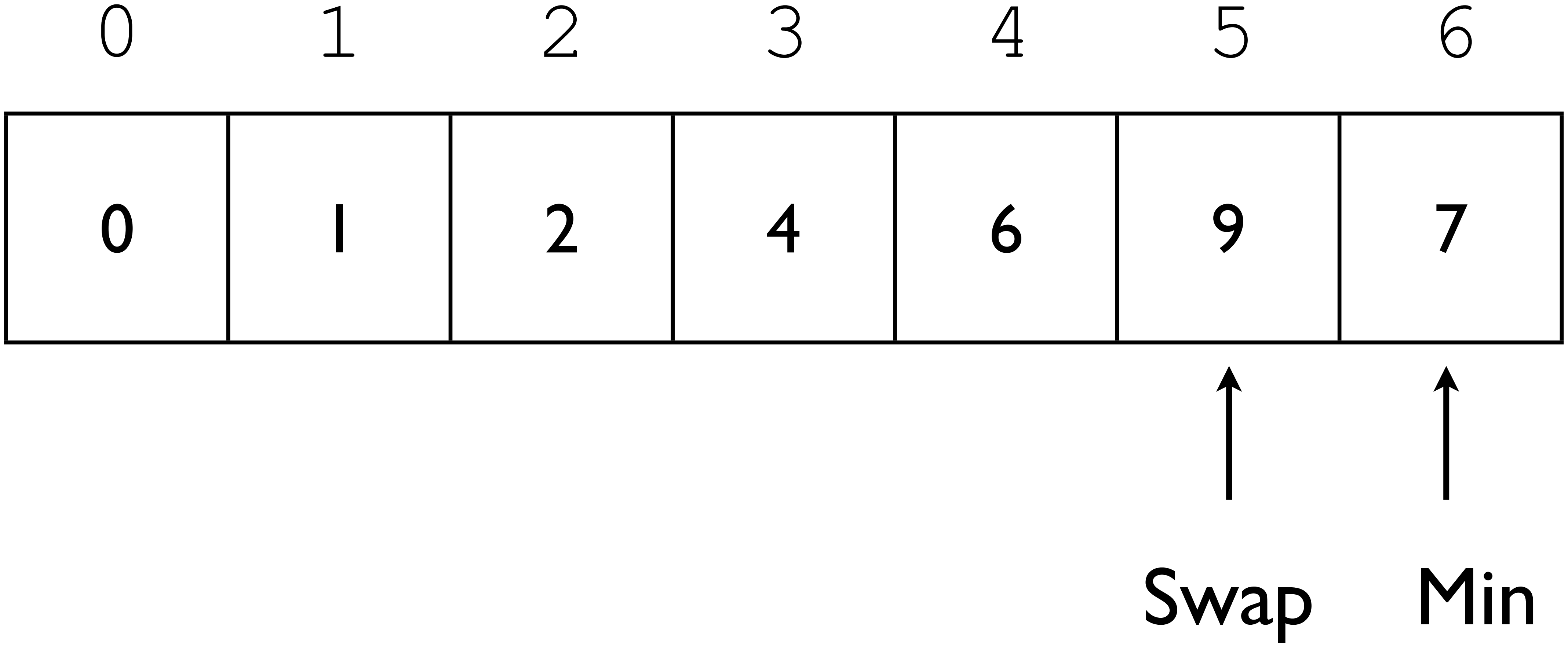


0	1	2	3	4	5	6
0	1	2	4	6	9	7

Swap Min

Selection Sort

Recorded Minimum Position: **6**
Recorded Minimum: **7**

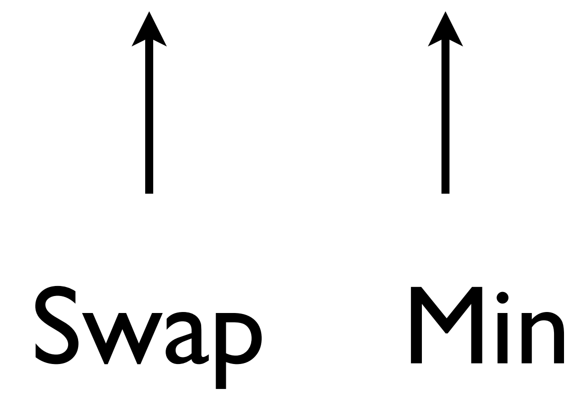


0	1	2	3	4	5	6
0	1	2	4	6	9	7

Selection Sort

Recorded Minimum Position: 6
Recorded Minimum: 7

0	1	2	3	4	5	6
0	**1**	**2**	**4**	**6**	**7**	**9**

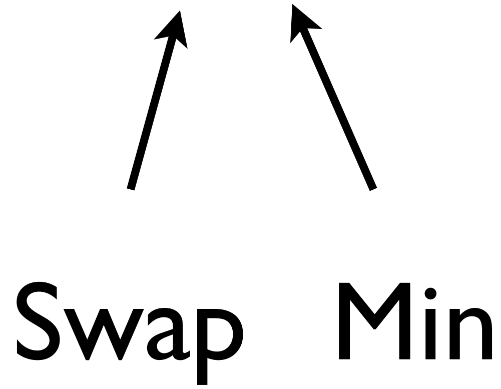


Selection Sort

Recorded Minimum Position: **6**

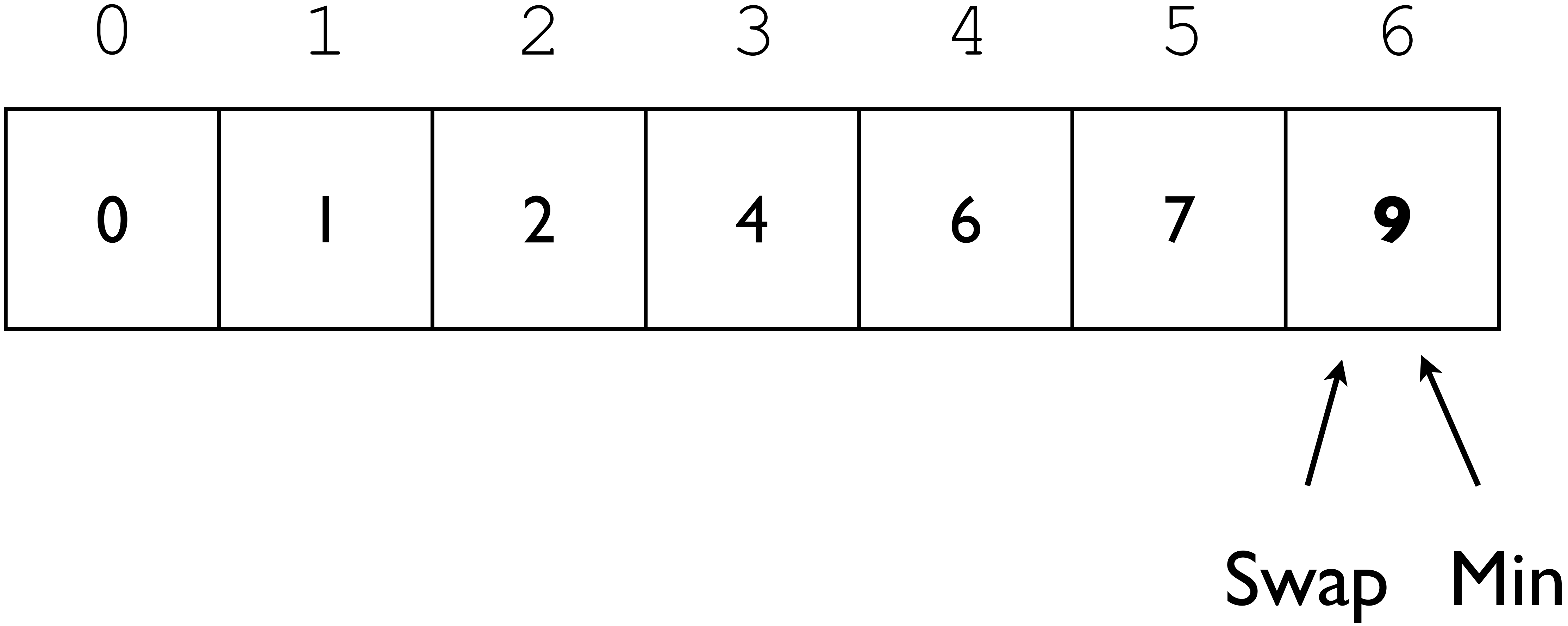
Recorded Minimum: **9**

0	1	2	3	4	5	6
0	**1**	**2**	**4**	**6**	**7**	**9**



Selection Sort

Recorded Minimum Position: 6
Recorded Minimum: 9



0	1	2	3	4	5	6
0	1	2	4	6	7	**9**

Swap Min

Code

Structs

Problem

- We want to represent a phone book
- Each entry has:
    - Name
    - Phone number
    - Address

Question

- Which type(s) is/are appropriate for:
  - Name?
  - Phone Number?
  - Address?

Possible Representation

- Use **parallel arrays**
  - Each array holds one kind of item
  - Index N refers to all information for entry #N

```
char** name;
char** address;
int* phoneNumber;
```

Problem

- Poor separation of concerns
- We have to pass around everything related to one person, which is annoying and error prone

```
void printPerson( char* name,
                  char* address,
                  int phone );
```

Another Solution

- Use structures, aka. `struct`s
- Put all data relevant to one entry in one place

```
struct person {
  char* name;
  char* address;
  int phone;
};
```

Structs

```
struct person {
   char* name;
   char* address;
   int phone;
};
```

```
void printPerson( struct person p );
```

Accessing Structs

- Use the dot (.) operator

```
struct person {
  char* name;
  char* address;
  int phone;
};

void printPerson( struct person p ) {
  printf( “Name: %s\n”, p.name );
  printf( “Address: %s\n”, p.address );
  printf( “Phone: %i\n”, p.phone );
}
```

Modifying Structs

- The dot (.) operator can be used along with assignment

```
struct person {
  char* name;
  char* address;
  int phone;
};
struct person p;
p.name = “foo”;
p.address = “123 Fake Street”;
p.phone = 0123456789
```

Pointers to Structs

- Structs can also be accessed via pointers
- Can access like so:

```
struct person p;
struct person* pointer = &p;
(*p).name = “foo”;
(*p).address = (*p).name;
(*p).phone = 0123456789
```

Pointers to Structs

- Structs can also be accessed via pointers
- Can also access with the more readable arrow operator

```
struct person p;
struct person* pointer = &p;
p->name = “foo”;
p->address = p->name;
p->phone = 0123456789
```

More on Structs (Only if Time Permits)

Struct Semantics

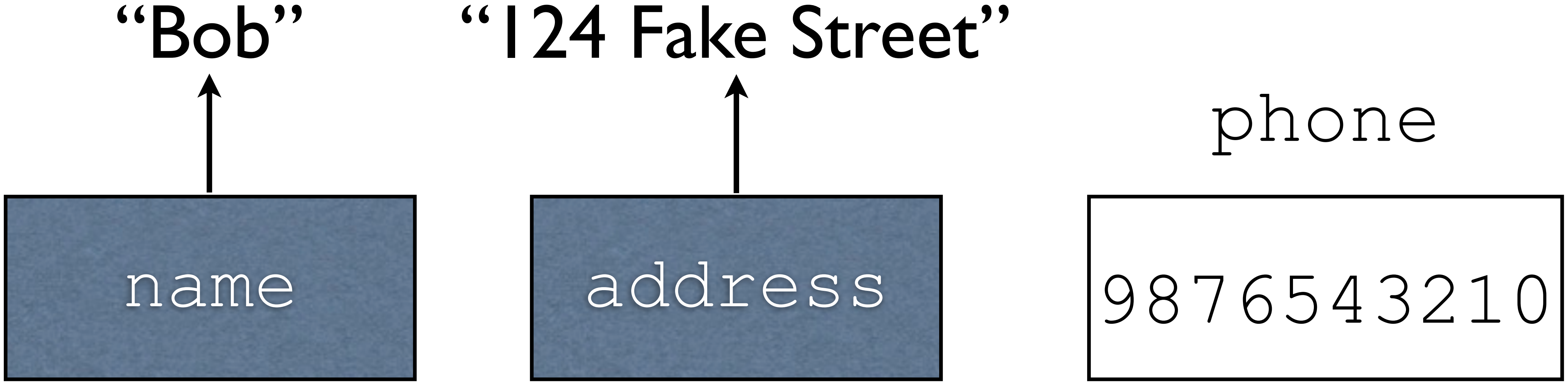
- Consider again:

```
void printPerson( struct person p );
```

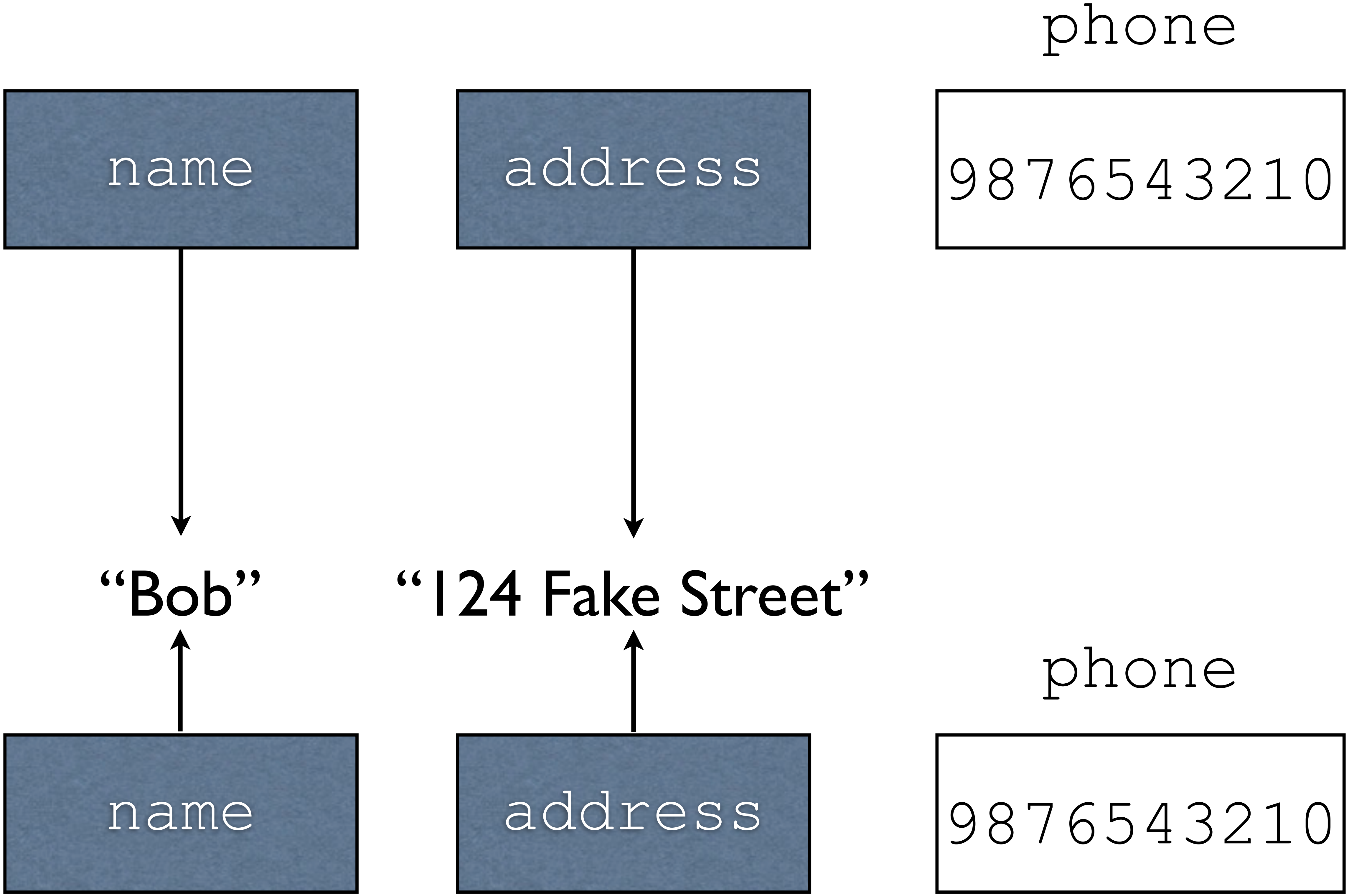
- When structs are passed, the whole thing is copied
- Note that this is a **shallow copy**

Shallow Copy

```
struct person {
   char* name;
   char* address;
   int phone;
};
```



Shallow Copy



phone

name

address

9876543210

“Bob”

“124 Fake Street”

phone

name

address

9876543210

Question

```
struct foo {
  int x;
};
void bar( struct foo f ) {
  f.x = 10;
}
int main() {
  struct foo f;
  f.x = 5;
  bar( f );
  // what’s f.x?
  return 0;
}
```

Question

```
struct foo {
  char* x;
};
void bar( struct foo f ) {
  f.x = “moo”;
}
int main() {
  struct foo f;
  f.x = “cow”;
  bar( f );
  // what’s f.x?
  return 0;
}
```

Structs and Pointers

- Oftentimes programmers will prefer pointers to structs as opposed to just structs
  - Avoids extra copying
  - **Possibly** appropriate