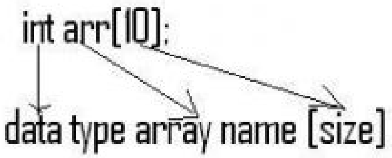
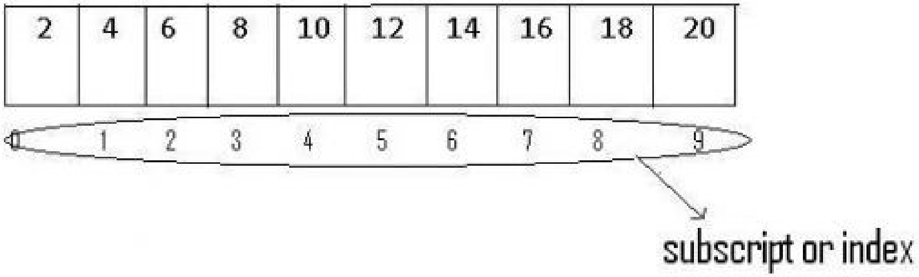
Intro to Arrays

Storing List of Data





2	4	6	8	10	12	14	16	18	20
0	1	2	3	4	5	6	7	8	9

Why Arrays

Suppose we want to store the grade for each student in a class

```
/* Need a variable for each? */
int bob, mary, tom, …;
```

Wow, cumbersome…

Easier to have a variable that stores the grades for all students

An *array* is a “Chunk of memory”

An array is a contiguous piece of memory that can contain multiple values

The values within the contiguous chunk can be addressed individually

Address in memory	0xeffffa00	0xeffffa04	0xeffffa08	0xeffffa0c	0xeffffa10	0xeffffa14	0xeffffa18	0xeffffa1c	0xeffffa20
grades	74	59	95	85	71	45	99	82	76

Array: “Chunk of memory”

Physical address	0xeffffa00	0xeffffa04	0xeffffa08	0xeffffa0c	0xeffffa10	0xeffffa14	0xeffffa18	0xeffffa1c	0xeffffa20
grades	74	59	95	85	71	45	99	82	76
index	0	1	2	3	4	5	6	7	8

Use an *index* to access individual elements of the array:
grades[0] is 74, grades[1] is 59, grades[2] is 95, and so on

Array Declaration

Syntax for *declaring* array variable:

```
type array_name[capacity];
```

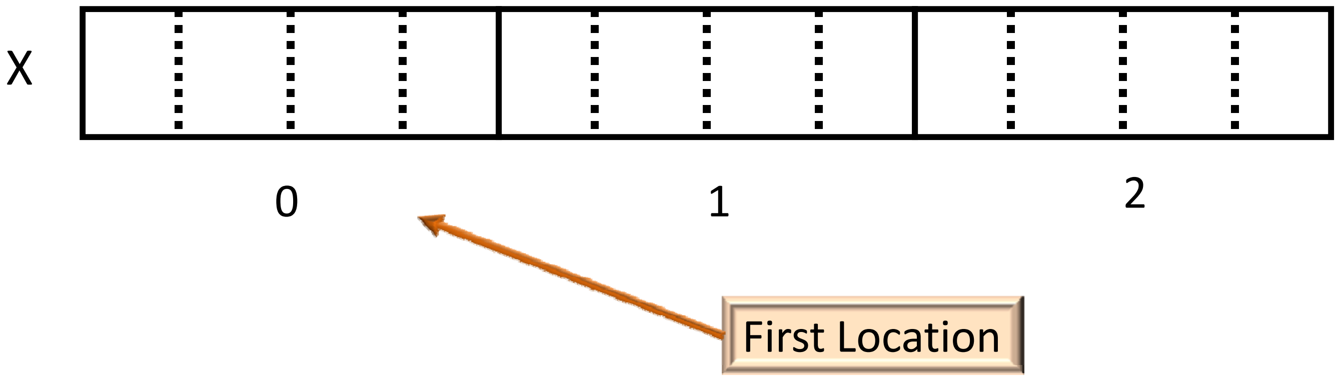
- type can be any type (int, float, char, …)
- array_name is an identifier
- capacity is the number of values it can store (indexing starts at 0)

Notice: The first location is location 0 (zero)!

Example

```
int x[3];        // an array of 3 integers
double y[7];  // an array of 7 doubles
```

Storage, e.g. 4-bytes per int



First Location

Operations with Arrays

- Assignment:
  - x[0] = 6;   /* Assign 6 to element x[0] */
  - y[2] = 3.1;   /* Assign 3.1 to element y[2] */

- Access
  - m = x[2];
  - p = y[0];

- Input/Output:
  - the elements are handled as their types, e.g.

```
scanf("%d %lf", &x[2], &y[3]);
printf("%d %lf\n",x[0], y[2]);    /* output 6 and 3.1 */
```

Arithmetic Operations

```
int main()
{
        double x[5];

        x[0] = 1;
        x[1] = 2;
        x[2] = x[0] + x[1];            /* X[2] = 3 */
        x[3] = x[2] / 3;               /* X[3] = 1 */
        x[4] = x[3] * x[2];            /* X[4] = 3 */
}
```

Variable Declaration for the array

`for` loops

“for” loops are ideal for processing elements in the array.

```
int main()
{
    int i;
    double values[4] = {3.14, 1.0, 2.61, 5.3};
    double sumValues = 0.0;

    for (i=0; i<4; i++)
    {
        sumValues = sumValues + values[i];
    }
    printf(“Sum = %lf\n”, sumValues);
}
```

`for` loops

“for” loops are ideal for processing elements in the array.

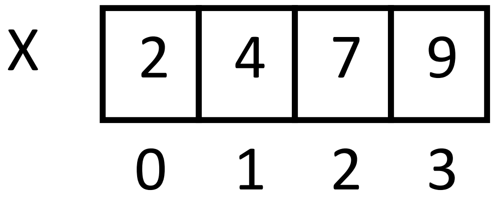
```
int main()
{
      int i;
      double values[4] = {3.14, 1.0, 2.61, 5.3};
      double sumValues = 0.0;

      for (i=0; i<=4; i++)
      {
            sumValues = sumValues + values[i];
      }
      printf(“Sum = %lf\n”, sumValues);
}
```

ERROR!
Out of bound

Initialization

Syntax: int X[4] = {2, 4, 7, 9};

Behavior: initialize elements starting with leftmost, i.e. element 0. Remaining elements are initialized to zero.



X	2	4	7	9
	0	1	2	3

Initialize all to 0: int X[4]={0};

Example

```
int main()
{
    double grades[5] = {90, 87, 65, 92, 100};
    double sum;
    int i;

    printf("The first grade is: %.1f\n", grades[0]);

    sum = 0;
    for(i=0;  i<5;  i++)
    {
        sum += grades[i];
    }
    printf("The average grade is: %.1f\n", sum / 5);

    grades[2] = 70;     /* Replaces 65 */
    grades[3] = grades[4];  /* Replaces 92 with 100 */
}
```

Constants for capacity

Good programming practice:
use #define for constants in your program

For example:

```
#define MaxLimit  25

int grades[MaxLimit];
for(int i; i<MaxLimit; i++){ };
```

If size needs to be changed, only the capacity "MaxLimit" needs to be changed.

Arrays as parameters of functions

```
int main()
{
    double values[4] = {3.14, 1.0, 2.61, 5.3};

    printf(“Sum = %lf\n”, SumValues( values, 4));
}
```

Suppose we want a function that sums up values of the array

Arrays as parameters of functions

```
double SumValues(double x[], int numElements)
{
    int i;
    double result = 0;
    for (i=0; i < numElements; i++)
        result = result + x[i];
    return result;
}
```

"[ ]" flags the parameter as an array.

- ALWAYS passed by reference

Array size is passed separately (as numElements)

Example

Program Behavior

1. Create an array of random numbers
2. Print unsorted array
3. Sort the array
4. Print sorted array

```
Array before sorting
  Element     0 :  58.7000
  Element     1 :   8.0100
  Element     2 :  72.3700
  Element     3 :   4.6500
  Element     4 :  58.3000
  Element     5 :  92.1700
  Element     6 :  95.3100
  Element     7 :   4.3100
  Element     8 :  68.0200
  Element     9 :  72.5400

Array after sorting
  Element     0 :   4.3100
  Element     1 :   4.6500
  Element     2 :   8.0100
  Element     3 :  58.3000
  Element     4 :  58.7000
  Element     5 :  68.0200
  Element     6 :  72.3700
  Element     7 :  72.5400
  Element     8 :  92.1700
  Element     9 :  95.3100
```

Sample output

The array elements are randomly generated

```
#include <stdio.h>
#include <stdlib.h>

void PrintArray( double [], int );
void SortArray( double [], int );
void Swap (double *, double *);
```

Functions are your friends! Make them work and then use them to do work!

```
#define NumElements 10

int main()
{
    int i;
    double values[NumElements]; /* The array of real numbers */

    srand(time(NULL));

    for (i=0; i < NumElements; i++)
    {
        values[i] = (double)(rand() % 10000) / 100.0;
    }

    printf("\nArray before sorting\n");
    PrintArray( values, NumElements );

    SortArray( values, NumElements );

    printf("\nArray after sorting\n");
    PrintArray( values, NumElements );

    return 0;
}
```

```
#define NumElements 10

int main()
{
    int i;
    double values[NumElements]; /* T

    srand(time(NULL));

    for (i=0; i < NumElements; i++)
    {
        values[i] = (double)(rand() % 10000) / 100.0;
    }

    printf("\nArray before sorting\n");
    PrintArray( values, NumElements );

    SortArray( values, NumElements );

    printf("\nArray after sorting\n");
    PrintArray( values, NumElements );

    return 0;
}
```

Array declaration

Declare an array of 10 doubles

The indices range from 0 to 9, i.e. `Value[0]` to `Value[9]`

```
#define NumElements 10

int main()
{
    int i;
    double values[NumElements]; /* The array of real numbers */

    srand(time(NULL));

    for (i=0; i < NumElements; i++)
    {
        values[i] = (double)(rand() % 10000) / 100.0;
    }

    print
    Print

    SortAr

    print
    Print

    retur
}
```

Initialize the array with random values

rand() returns a pseudo random number between 0 and RAND_MAX

rand()%10000 yields a four-digit integer remainder

/100.0 moves the decimal point left 2 places

So, Values is an array of randomly generated 2-decimal digit numbers between 0.00 and 99.99

```
printf("\nArray before sorting\n");
PrintArray( values, NumElements );
```

PrintArray prints the elements of the array in the order they are given to it

```
SortArray( values, NumElements );
```

SortArray sorts the elements into ascending order

```
printf("\nArray after sorting\n");
PrintArray( values, NumElements );
```

Parameter Passing

```
void PrintArray( double array[], int size )
{
}
```

array is a C array of doubles
array is passed by reference, i.e. any changes to parameter *array* in the function would change the argument values
The array size is passed as "size"

```
void PrintArray( double array[], int size )
{
    int i;

    for (i=0; i<size; i++)
      printf("  Element %5d : %8.4lf\n",i, array[i]);
}
```

array[i] is a double so the output needs to be “%f”

The range of the “for” statement walks through the whole array from element 0 to element N-1.

Sorting Array

```
void SortArray( double array[], int size)
{
}
```

array is an array of doubles.

array is passed by reference, i.e. changes to parameter array change the argument values There is no size restriction on array so the size is passed as “size”.

Selection Sort

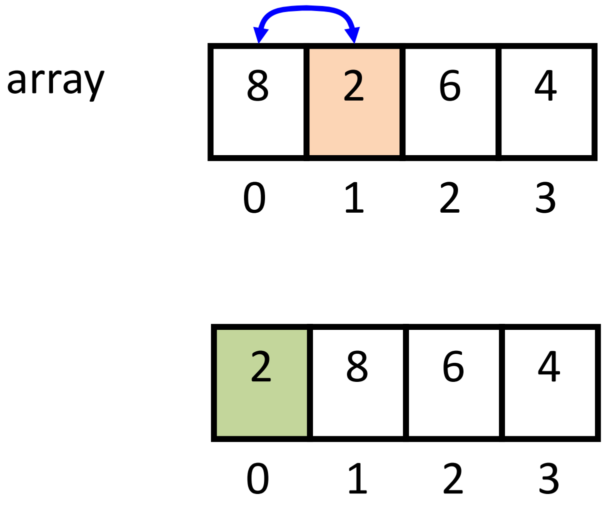
array	8	2	6	4
	0	1	2	3

Selection Sort

array	8	2	6	4
	0	1	2	3

Search from array[0] to array[3]
to find the smallest number

Selection Sort



Search from array[0] to array[3] to find the smallest number and swap it with array[0]

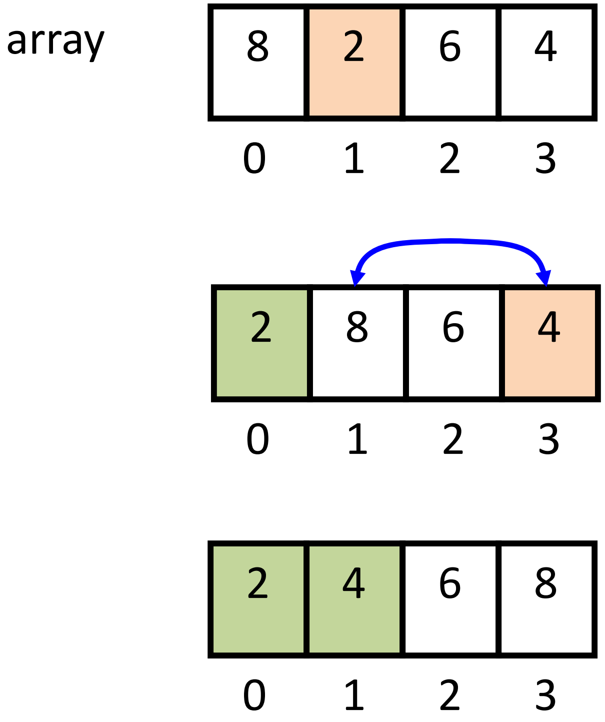
Selection Sort

array	8	2	6	4
	0	1	2	3

2	8	6	4
0	1	2	3

Search from array[1] to array[3]
to find the smallest number

Selection Sort



array

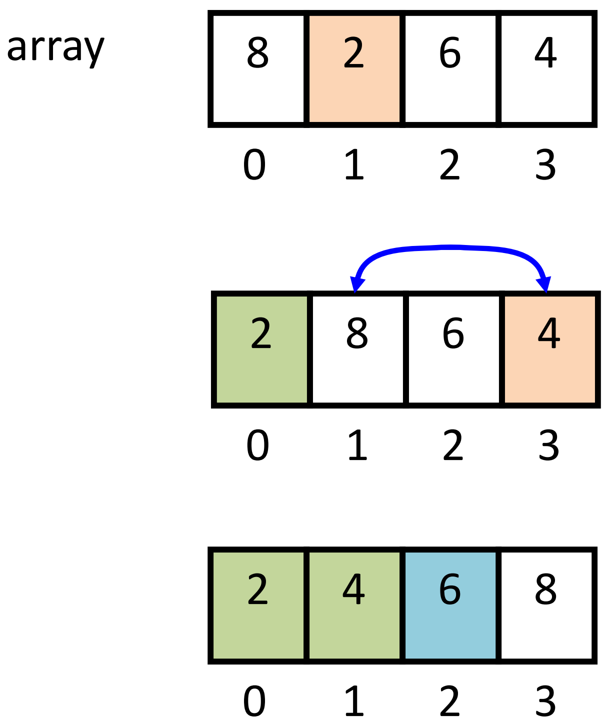
8	2	6	4
0	1	2	3

2	8	6	4
0	1	2	3

2	4	6	8
0	1	2	3

Search from array[1] to array[3] to find the smallest number and swap it with array[1]

Selection Sort



array

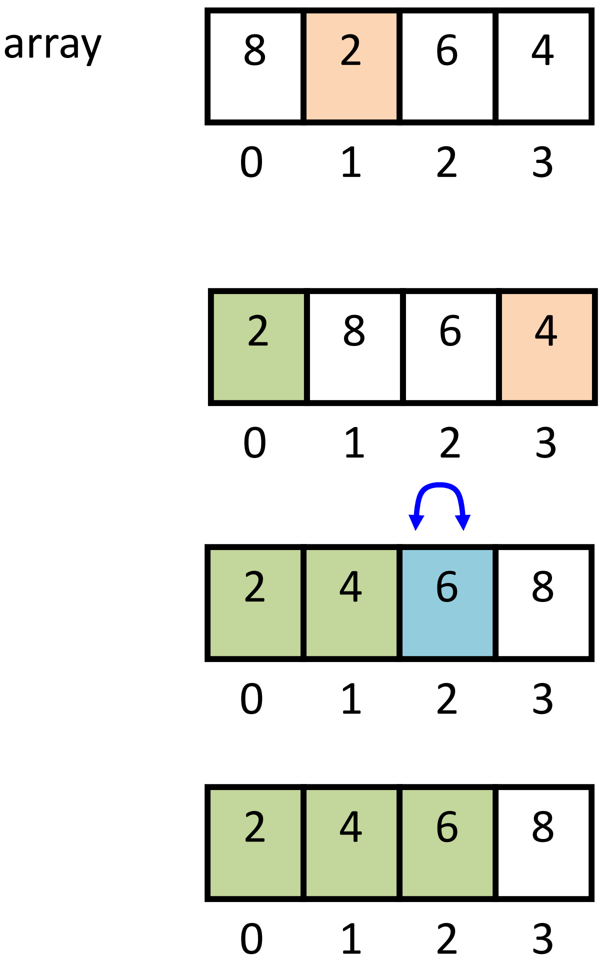
8	2	6	4
0	1	2	3

2	8	6	4
0	1	2	3

2	4	6	8
0	1	2	3

Search from array[2] to array[3] to find the smallest number and swap it with array[2]

Selection Sort



array

8	2	6	4
0	1	2	3

2	8	6	4
0	1	2	3

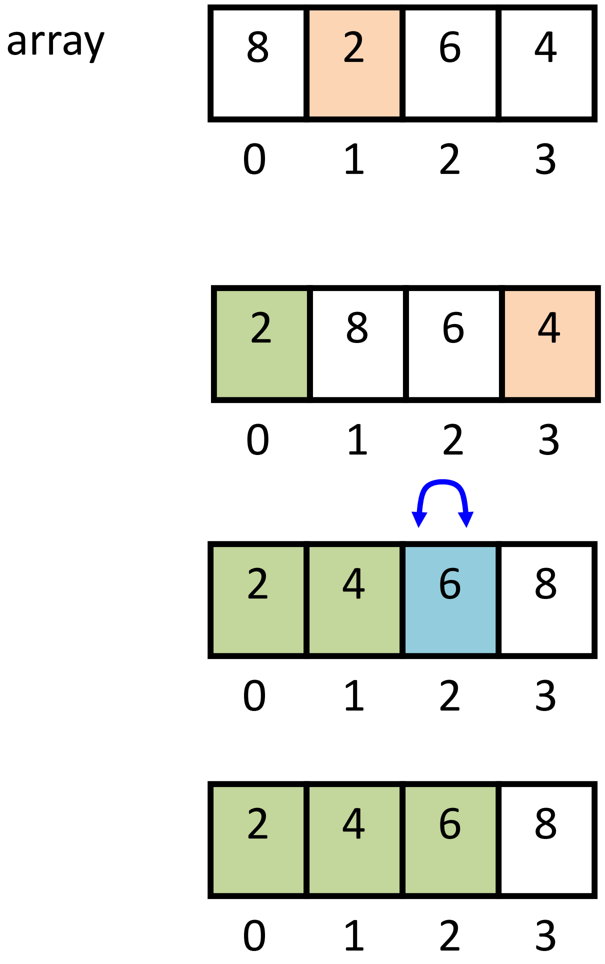
2	4	6	8
0	1	2	3

2	4	6	8
0	1	2	3

Search from array[2] to array[3] to find the smallest number and swap it with array[2]

And we are done!

Selection Sort



array

8	2	6	4
0	1	2	3

2	8	6	4
0	1	2	3

2	4	6	8
0	1	2	3

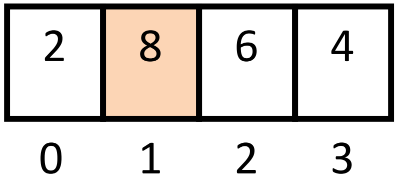
2	4	6	8
0	1	2	3

How many iterations are there?

Answer: 3 (from i = 0 to i = 2)

More generally, if number of elements in the array is size, you need to iterate from i = 0 to i = size - 2

Selection Sort

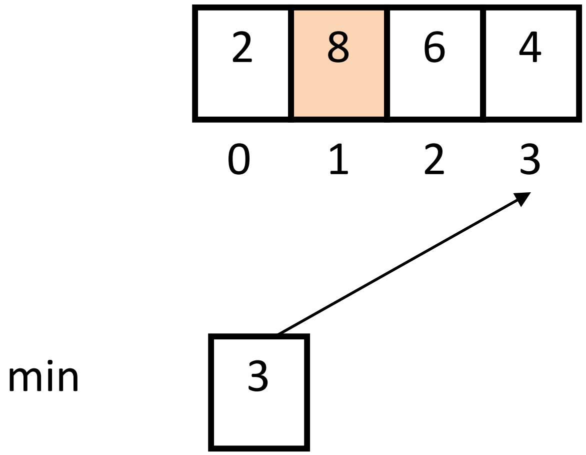


2	8	6	4
0	1	2	3

At every iteration i, you need to search from array[i] to array[size – 1] to find the smallest element

How to do this?

Selection Sort

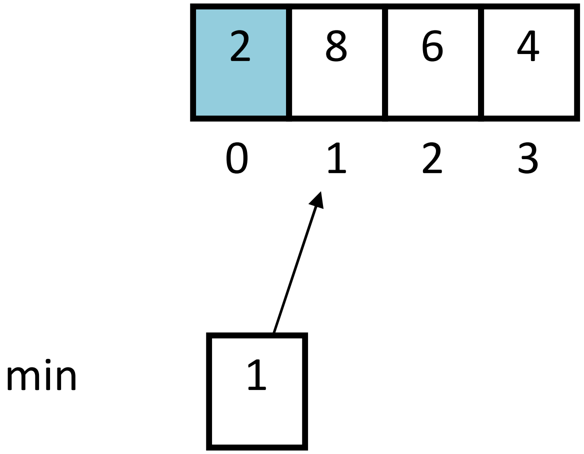


At every iteration i, you need to search from array[i] to array[size – 1] to find the smallest element

How to do this?

Use a variable called min to locate the *index* of the smallest element

Selection Sort



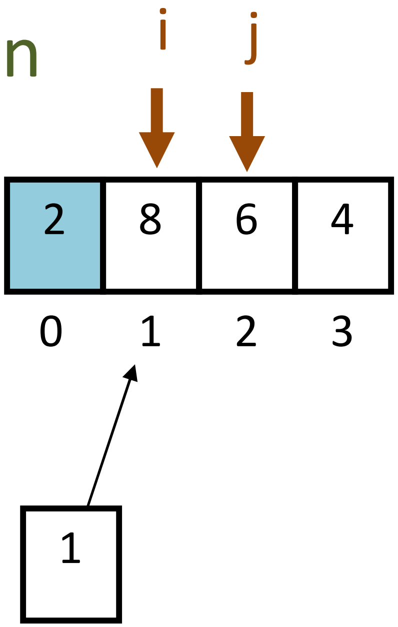
2	8	6	4
0	1	2	3

min 1

Assume current iteration i = 1

Initialize min = i

Selection Sort



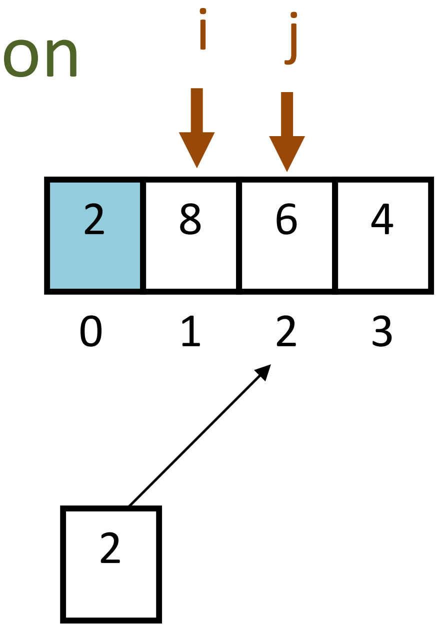
Assume current iteration i = 1

Initialize min = I

Set j = i + 1

Compare array(min) to array(j)

Selection Sort



min

Assume current iteration i = 1

Initialize min = i

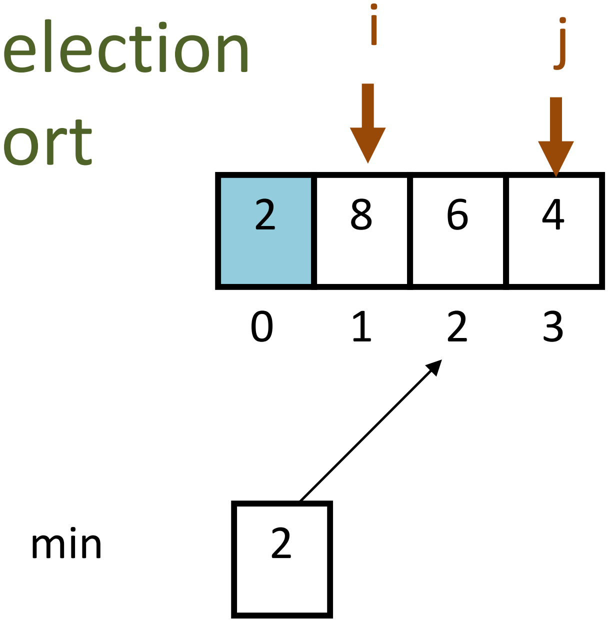
Set j = i + 1

Compare array(min) to array(j)

If array(j) < array(min)
    set min to j

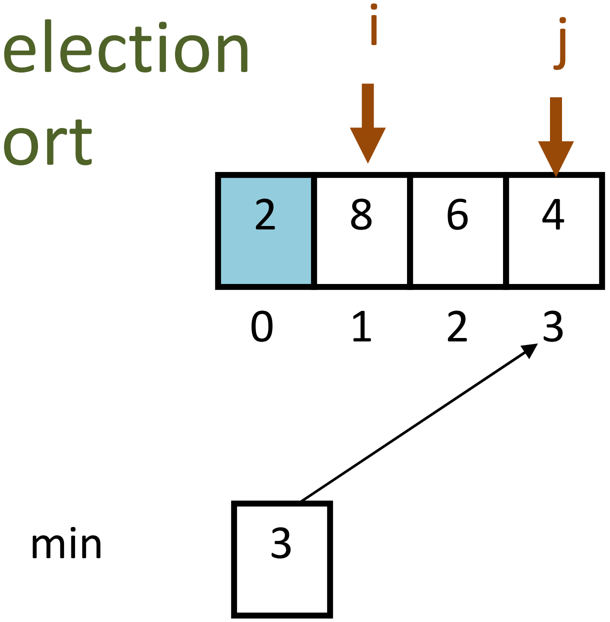
Because 6 < 8,
    min is now set to 2

Selection Sort



Increment j
Compare array(min) to array(j)

Selection Sort



Increment j

Compare array(min) to array(j)

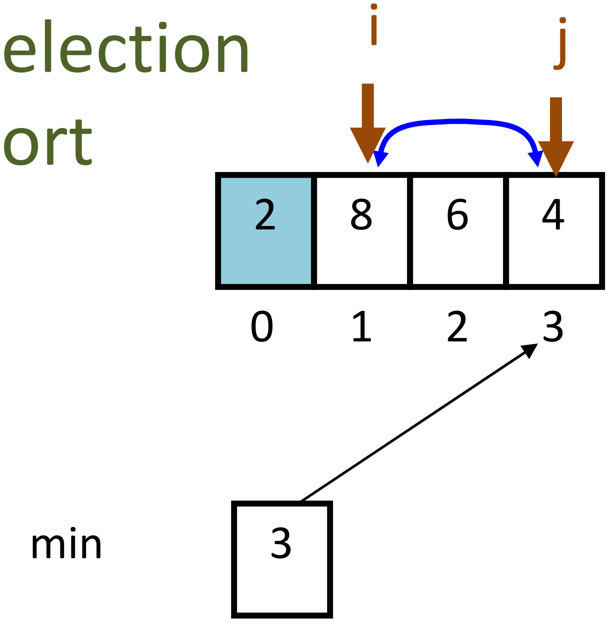
If array(j) < array(min)

set min to j

Because 4 < 6,

min is now set to 3

Selection Sort



Swap array(i) with array(min)

SortArray

```
void SortArray( double array[], int size)
{
    int i, j, min;

    for (i=0; i < size-1; i++)
    {
        min = i;
        for (j=i+1; j<size; j++)
        {
            if (array[j] < array[min])
            {
                min = j;
            }
        }

        Swap(&array[i], &array[min]);
    }
}
```

Swap

```
void Swap (double *a, double *b)
{
   double temp = *a;
   *a = *b;
   *b = temp;
}
```

Swap

```
void Swap (double *a, double *b)
{
   double temp = *a;
   *a = *b;
   *b = temp;
}
```

Note: We're passing two elements of the array; not passing the entire array

So, we **CANNOT** declare it as

```
void Swap(double a, double b)

void Swap(double a[], double b[])
```