COMP10002 Workshop Week 5

Array & Pointer Madness

MST: 2PM this Friday, in person, on paper!

1. Complexity
2. Arrays
3. Using library arrayops.h in Ed for exercises in chapter 7
4. Discuss 7.4, 7.6, 7.7

LAB:

- “on paper” coding: 7.04, 7.06, 7.07 and others

Topics in lec05.pdf are important in the MST– Questions?

- pp. 1-7: Array Fundamental:
  - Concepts (pp. 1-3)
  - Arrays – Pointers (p. 4)
  - Functions with Arrays as arguments (p. 4)
  - examples & exercises (pp. 5-6)
  - summary (p. 7)

- p.8: Algorithms (concept)
- pp. 16-22: Algorithm Complexity, Defining *Efficiency* with *Big-O* :
  - pp. 16-18: defining efficiency, big-O definition
  - pp. 19-20: example finding big-O for some f(n)
  - p. 21: keep O(f(n)) simple by dropping constants and lower terms
  - p. 22: why using big-O

Algorithm Efficiency

- Efficiency of an algorithm is expressed as a function T(n) of input size n
- T(n) represents the number of computation steps in the worst case of the algorithm
- We use Big-O Notation to reduce T(n) to an efficiency class such as $O(\log n)$, $O(n)$, $O(n^2)$.
- $T(n) \in O(f(n))$ if, except for a constant factor, T(n) *grows slower or as the same rate* as f(n) for all large n.
- Big-O Notation is about asymptotic behaviour. That implies the rules:

Rule	Examples
Drop constant factor	$9n^2 \in O(n^2)$ $\quad$ $1000 n \log n \in O(n \log n)$
Drop lower-order terms	$9n^2 + 10000n + 10^9 \in O(9n^2) \in O(n^2)$
base of log is not important	$\log_a n \in O(\log_b n) \in O(\log n)$

Algorithm Efficiency in Practice, supposing that n represents the input size

Example 1	Example 2	Big-O
a= b;	if (a>b+c*d) a= b + c*d/a;	O(1)

for (i=0; i<n; i++) {
  // a O(1) stuff
}
	for (i=0; i<3*n+7; i++) {
	  // a O(1) stuff
	}
		O(?)

for (i=0; i<n; i++) {
 for (j=0; j<n; j++) {
  // a O(1) stuff
 }
}
	for (i=0; i<n; i++) {
	 for (j=i; j<n; j++) {
	  // a O(1) stuff
	 }
	}
		O(?)

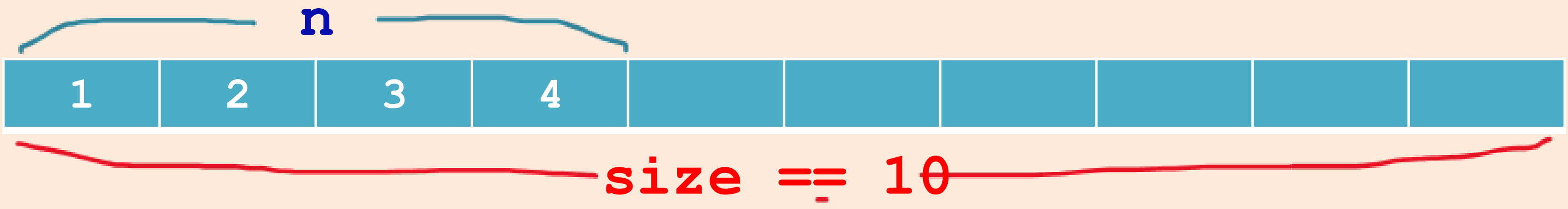
for (i=1; i<n; i *= 2) {
 // a O(1) stuff
}
	for (i=n; i>0; i /= 3) {
	 // a O(1) stuff
	}
		O(?)

Arrays = ?

- Array= a collection of the same datatype's containers
  = storing a sequence of the same datatype's values

- When using an array, we need 4 stuffs:
  - data type of each container
  - name of array
  - size (ie. capacity)
  - length (ie. current number of elements)

```
int A[10]={1,2,3,4}, n=4;
```



Array in computer memory

- Array occupies a block of "size" adjacent cells in memory

arrays: a collection of variables under a common name

	statements	in memory (*after* running LHS)					
1	`int i, A[5];`	**i**	**A[0]**	**A[1]**	**A[2]**	**A[3]**	**A[4]**
2	`A[0] = 10;` `i= A[0] * 2;`	20	10				
3	`i= 2;` `A[i]= 20;`	2	10		20		

4	`for (i=0; i<5; i++) {`
	`    A[i]= i*i;`
	`}`
		5	0	1	4	9	16

5	`for (i=0; i<3; i++) {`
	`    scanf ( "%d", &A[i] );`
	`} /*` supposing input `10 20 30 */`						

arrays...

	statements	variables in memory
1	int i, sum=0, A[5]= {0,1,2,3,4};	i sum A[0] A[1] A[2] A[3] A[4] 0 0 1 2 3 4

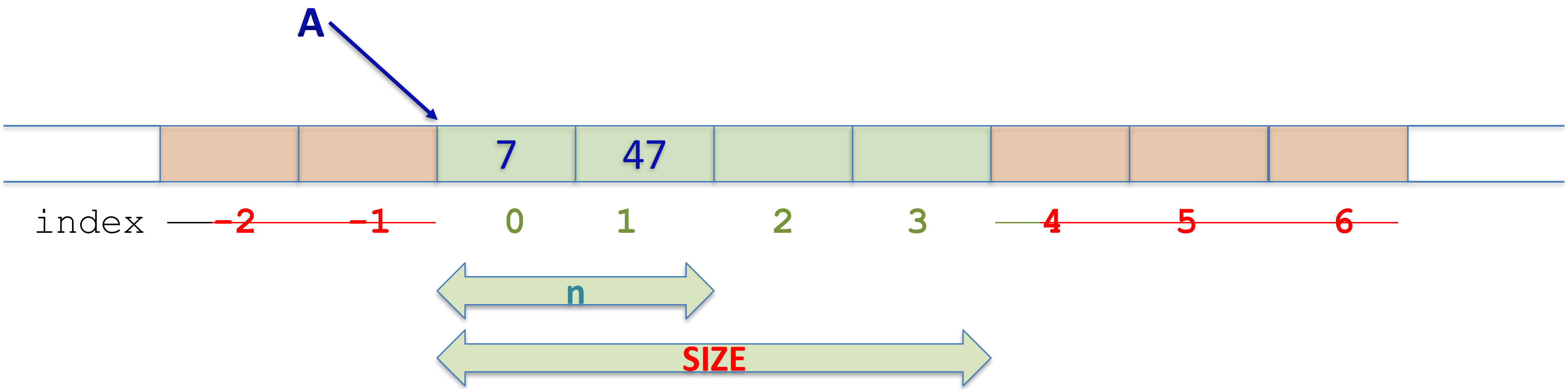
2	for (i=0; i<5; i++){
	A[i] *= 3
	sum += A[i];
	}
		5 30 0 3 6 9 12

3	for (i=0; i<4; i++) {
	A[i+1]= A[i];
	}	

Chorus of the day: Array names are pointer constants!

```
#define SIZE 4
int A[SIZE]= {007,47};  // A is a pointer,  A has the value of &A[0]
int n= 2;
```

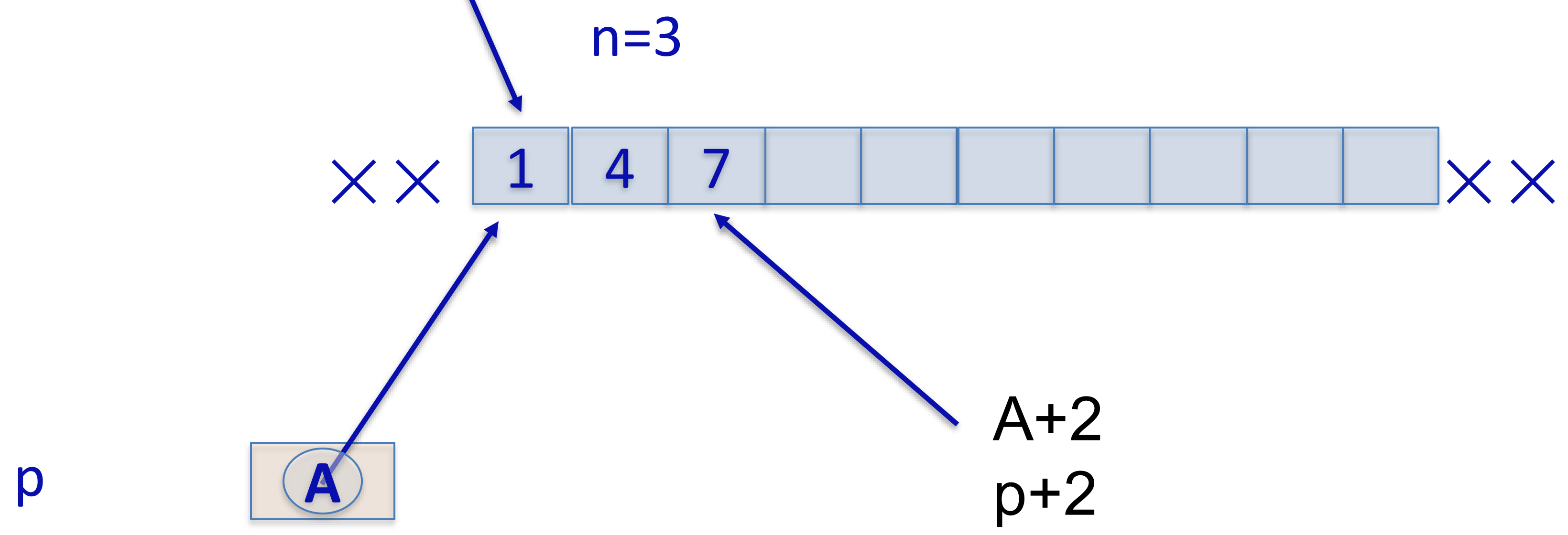
- the system keeps track of the starting address A, but NOT the SIZE 4,
- you *can*, but should not use A[-1], A[4] or A[5] – they are **undefined** for the array and their use will lead to a disaster!



Chorus of the day: Array names are pointer constants!
if p points to me, p+1 points to the next guy

```
#define SIZE 10t
  int    A[SIZE]= {1,4,7}; int n= 3;

int *p;        // p is a variable pointer
p=  ;A         // p now can be used as an alias for A
          // p[i] is the same as A[i]
```



In this context:

- A is the same as &A[0]
- p+1 points to the memory location immediately following the one pointed to by p

Passing an Array to a Function= passing array name and array length

Example of function

```
int foo(int A[], int n){
  int sum= 0, i;
  for (i=0; i<n; i++) {
    sum += A[i];
  }
  for (i=0; i<n; i++) {
    A[i]= 0;
  }
  return sum;
}
```

Example: using function

```
int B[]={1,2,3}, s;
s= foo(B, 3);
```

Pass:

- the array's name (the start)
- the array's length

What function foo can do with the call foo(B, 3) ?

uses value 3 and array B[]

modifies the elements of array B[] (via A[])

Check your understanding

```
int sum(int A[], int n){
  int i, s= 0;
  for (i=0; i<n; i++) {
    s += A[i];
  }
  return s;
}
```

With the declarations:

```
int B[10]= { 0,1,2,3,4,5,6,7,8,9 };
```

For each of the following statements:

- is it valid?
- if yes, what's the output?

```
printf("%d\n", sum(B, 10));
printf("%d\n", sum(B+4, 2));
printf("%d\n", sum(&B[0], 5));
printf("%d\n", sum(B+0, 5));
printf("%d\n", sum(B+3, 8));
```

Example: Design Ex 7.04. Also make sure how to start the implementation.

7.04: *Write a program that reads as many as 1,000 integer values, and counts the frequency of each value in the input:*

```
./program
Enter as many as 1000 values, ^D to end
1 1 1 3 3 3 3 3 4 6 4 3 6 10 3 5 4
17 values read into array
Value Freq
  1   3
  3   7
  4   3
  5   1
  6   2
 10   1
```

- Input= ? , Output= ?
- Approach= ?
- Know how to test this function in Exercise 7.04, and hence
- Know how to use the library arrayops.h in this and other array exercises

Sorting

The Task:

Input: Given an array of n elements.

Output: The same input array, but the its elements are re-arranged in some *order* (such as *increasing*)

Example: sort array in non-decreasing order)

Input : `int A[5]= {7,2,5,5,6}, n= 5;`

Output: `A[]` becomes `{2,5,5,6,7}`

Algorithms:

- Insertion Sort (see animate-insertionsort.pdf)
- Selection Sort (Ex.7.06)
- Quick Sort (next week)
- …

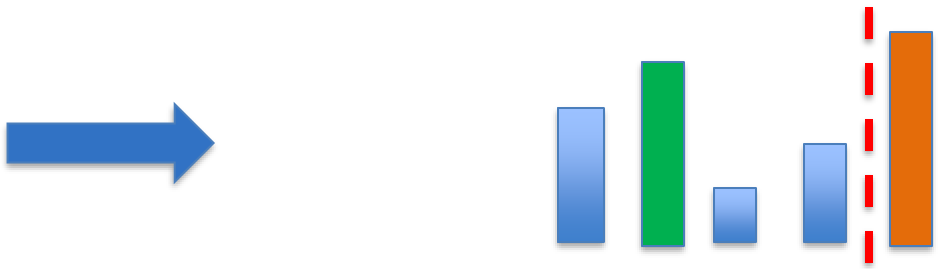
Ex 7.06: Selection Sort - understanding

<u>Selection sort</u>: scan the array to determine the location of the largest element, and swap it into the last position. Then repeat the process, concentrating at each stage on the elements that have not yet been swapped into their final position.

*Round **1**:*

scan all un-sorted elements from position `0` *to position* `n-1` *to determine the position of the largest element,*

and swap it into the last position, ie. position `n-1`*.*



What next?

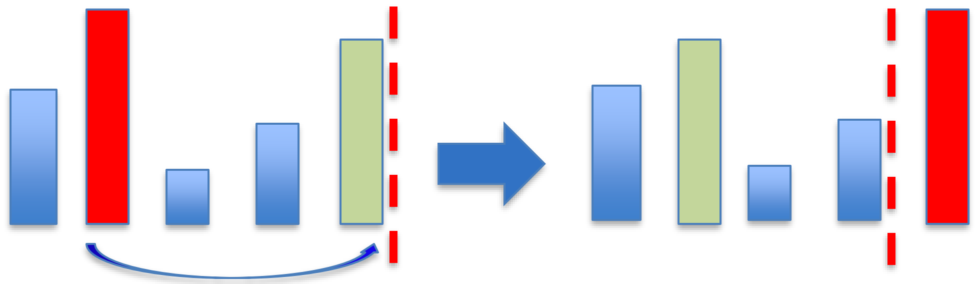
How many rounds?

*What is the aim of round number **i** ?*

Ex 7.06: Write A Function (and only write the function, as required in the MST)

to sort A[0..n-1]
- *first, swap* A[n-1] *with the largest of* A[0..n-1]
- *then, swap* A[n-2] *with the largest of* A[0..n-2]

- *last, swap* A[1] *with the largest of* A[0..1]



```
//  sort array A in ascending order using selection sort
void sel_sort(int A[], int n) {

}
```

Discuss 7.07: Most frequent element

Input :

Output :

Approach:

7.07: Most frequent element

Input: an array of n integers

Output:

- the value in A that appears most frequently: {2,1,3,1} →
- If there is a tie, return the smallest such value: {3,3,2,2} →

Approach:

Today's Main Topics

- Big-O in practice.
- Writing array functions (on paper!)

Q&A

- Precedence order of operators
- Efficiency
- MST sample
- Writing functions with arrays
- other topics in chapters 1 - 7

next week:

- Linear Search
- Binary Search
- 2D arrays

Algorithms are Fun,
and so should be the MST

Today's Lab Time:

Write functions on paper!

or, better

write on whiteboard with a friend

Mandatory:

Ex7.04: Counts frequency of each value in the array
Ex7.06: Selection sort,
Ex7.07: Most frequent element

Additional:

Ex7.03: sorts the array then removes duplicates
Ex7.x1: Is the array sorted?
Ex7.x2: Count distinct values

Likely Next Week:

Ex7.08: k-th smallest
Ex7.09: count ascending runs
Ex7.10: count inversions

Additional Slides

Quiz Discussion

Given function:

```
int foo(int A[], int n) {
  int i, sum= 0;
  for (i=0; i<n; i++) {
     sum += A[i];
     A[i] *= 2;
  }
  return sum;
}
```

Quiz 1: What is the output of:

```
int A[10]= {0,1,2,3,4,5,6,7};
printf("sum= %d, A[5]=%d\n",
   foo(A,10), A[5]);
```

A	sum= 28, A[5]= 5
B	sum= 28, A[5]= 4
C	sum= 28, A[5]= 10
D	sum= 56, A[5]= 8
E	syntax and/or logic errors

Quiz 2: What is the output of:

```
int A[10]= {0,1,2,3,4,5,6,7};
printf("sum= %d, A[7]=%d\n",
       foo(A+2,4), A[7]);
```

A	sum= 14, A[7]= 7
B	sum= 15, A[7]= 14
C	sum= 28, A[7]= 14
D	sum= 10, A[7]= 7
E	syntax and/or logic errors