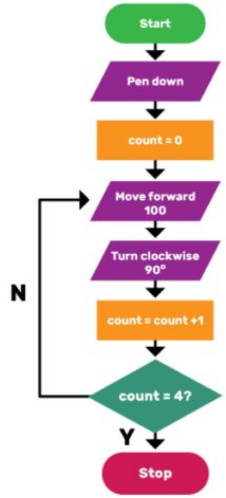
Introduction to Algorithm

"Move forward 100 pixels, then turn clockwise 90 degrees. Do this a total of 4 times."

```
pen Down
count <- 0

repeat until count = 4
  forward 100
  clockwise 90°
  count = count +1
```



What is an algorithm?

- An algorithm is "**a finite set of precise instructions for performing a computation or for solving a problem**"

- A program is one type of algorithm
  - All programs are algorithms
  - Not all algorithms are programs!

- Design a scheduler for RTOS is an algorithm

Some algorithms are harder than others

- Some algorithms are easy
  - Finding the largest (or smallest) value in a list
  - Finding a specific value in a list
- Some algorithms are a bit harder
  - Sorting a list
- Some algorithms are very hard
  - Finding the shortest path between Miami and Seattle
- Some algorithms are essentially impossible
  - Factoring large composite numbers

☐ Algorithm complexity needs to be considered

Algorithm 1: Maximum Element

- Given a list, how do we find the maximum element in the list?
- To express the algorithm, **Pseudocode** can be used

```
procedure max(a1, a2, …, an: integers)
max := a_1
for i := 2 to n
    if max < a_i then max := a_i
```

Maximum element running time

- How long does this take?

- If the list has *n* elements, worst case scenario is that it takes *n* "steps"
    - Here, a step is considered a single step through the list

Properties of Algorithms

- Algorithms generally share a set of properties:
  - Input: what the algorithm takes in as input
  - Output: what the algorithm produces as output
  - Definiteness: the steps are defined precisely
  - Correctness: should produce the correct output
  - Finiteness: the steps required should be finite
  - Effectiveness: each step must be able to be performed in a finite amount of time
  - Generality: the algorithm *should* be applicable to all problems of a similar form

Searching Algorithms

- Given a list, find a specific element in the list

- We will see two types
  - Linear search
    - a.k.a. sequential search
  - Binary search

Algorithm 2: Linear Search

- Given a list, find a specific element in the list
  - List does NOT have to be sorted!

procedure linear_search (x: integer; $a_1, a_2, \ldots, a_n$: integers)

$i := 1$

while ($i \leq n$ and $x \neq a_i$)

$i := i + 1$

if $i \leq n$ **then** *location* := i

else *location* := 0

{*location* is the subscript of the term that equals x, or it is 0 if x is not found}

Linear Search Running Time

- How long does this take?

- If the list has *n* elements, worst case scenario is that it takes *n* "steps"
    - Here, a step is considered a single step through the list

- Complexity is O(N)

Algorithm 3: Binary Search

- Given a list, find a specific element in the list
  - List MUST be sorted!
- Each time it iterates through, it cuts the list in half

procedure binary_search (x: integer; $a_1, a_2, \ldots, a_n$: increasing integers)

i := 1 { i is left endpoint of search interval }

j := n { j is right endpoint of search interval }

while $i < j$

begin

m := $\lfloor (i+j)/2 \rfloor$ { m is the point in the middle }

if x > a_m **then** i := m+1

else j := m

end

if $x = a_i$ **then** *location* := i

else *location* := 0

Binary Search Running Time

- How long does this take (worst case)?

If the list has 8 elements
  - It takes 3 steps

- If the list has 16 elements
  - It takes 4 steps

- If the list has *n* elements
  - It takes $\log_2 n$ steps

Sorting Algorithms

- Given a list, put it into some order
  - Numerical, lexicographic, etc.

- We will see two types
  - Bubble sort
  - Insertion sort

Algorithm 4: Bubble Sort

- One of the most simple sorting algorithms
  - Also one of the least efficient
- It takes successive elements and "bubbles" them up the list

procedure bubble_sort (a_1, a_2, …, a_n)
for i := 1 **to** n-1
    for j := 1 **to** n-i
    if $a_j > a_j$+1
    then interchange a_j and a_j+1
{ a_1, …, a_n are in increasing order }

Bubble Sort Running Time

- Outer for loop does n-1 iterations
- Inner for loop does
  - n-1 iterations the first time
  - n-2 iterations the second time
  - …
  - 1 iteration the last time
- Total: $(n-1) + (n-2) + (n-3) + \ldots + 2 + 1 = (n^2-n)/2$
  - We can say that's "about" ***n^2* time**

Algorithm 5: Insertion Sort

- Another simple (and inefficient) algorithm
- It starts with a list with one element, and inserts new elements into their proper place in the sorted part of the list

procedure insertion_sort ($a_1, a_2, \ldots, a_n$)
for j := 2 **to** n
begin
i := 1
while $a_j > a_i$
$i := i + 1$
$m := a_j$
for k := 0 **to** j-i-1
$a_{j-k} := a_{j-k-1}$
$a_i := m$
end { $a_1, a_2, \ldots, a_n$ are sorted }

Insertion Sort Running Time

- Outer for loop runs n-1 times
- In the inner for loop:
  - Worst case is when the while keeps i at 1, and the for loop runs lots of times
  - If i is 1, the inner for loop runs 1 time (k goes from 0 to 0) on the first iteration, 1 time on the second, up to n-2 times on the last iteration
- Total is 1 + 2 + … + n-2 = (n-1)(n-2)/2
  - We can say that's "about" $\boldsymbol{n^2}$ **time**

Comparison of Running Times

- Searches
  - Linear: *n* steps
  - Binary: $\log_2 n$ steps
  - Binary search is about as fast as you can get
- Sorts
  - Bubble: n^2 steps
  - Insertion: n^2 steps
  - There are other, more efficient, sorting techniques
    - In principle, the fastest are heap sort, quick sort, and merge sort
    - These each take take $n * \log_2 n$ steps
    - In practice, quick sort is the fastest, followed by merge sort

RTOS 'update' function

```
void SCH_Update(void)  {
    tByte Index;
    // NOTE: calculations are in *TICKS* (not milliseconds)
    for (Index = 0; Index < SCH_MAX_TASKS; Index++) {
    // Check if there is a task at this location
        if (SCH_tasks_G[Index].pTask) {
      if (SCH_tasks_G[Index].Delay == 0) {
      // The task is due to run
            SCH_tasks_G[Index].RunMe += 1; // Inc. the 'RunMe' flag
            if (SCH_tasks_G[Index].Period) {
                // Schedule periodic tasks to run again
                SCH_tasks_G[Index].Delay = SCH_tasks_G[Index].Period;
            }
      } else {
            // Not yet ready to run: just decrement the delay
            SCH_tasks_G[Index].Delay -= 1;
      }
        }
    }
}
```

MCQ

The word _______________comes from the name of a Persian mathematician

a) Flowchart
b) Flow
c) Algorithm
d) Syntax

MCQ

The time that depends on the input: an already sorted sequence that is easier to sort.

a) Process
b) Evaluation
c) Running
d) Input

MCQ

- Algorithms can be represented (select incorrect):
  a) as pseudo codes
  b) as syntax
  c) as programs
  d) as flowcharts

MCQ

- When an algorithm is written in the form of a programming language, it becomes a __________

a) Flowchart
b) Program
c) Pseudo code
d) Syntax

MCQ

- Any algorithm is a program.

a) True
b) False

Any program is an algorithm

- a) True
  b) False

MCQ

A system wherein items are added from one and removed from the other end

a) Stack
b) Queue
c) Linked List
d) Array

MCQ

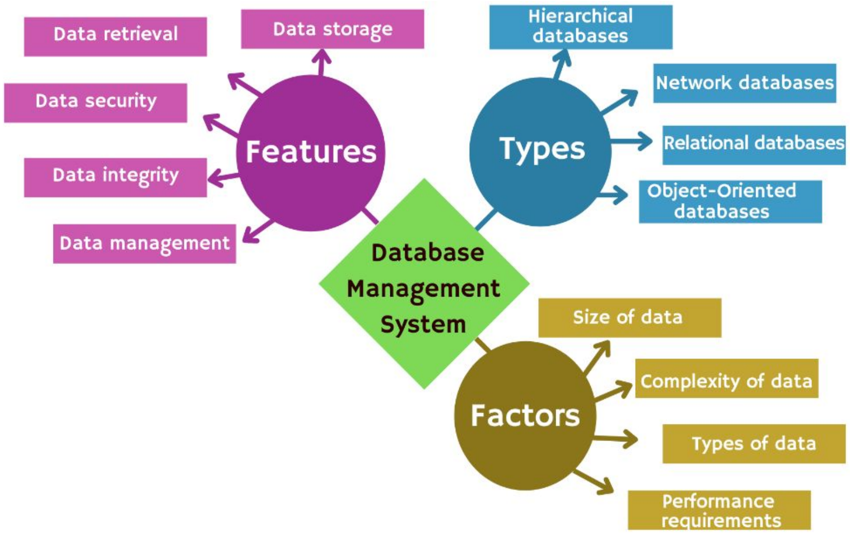
Another name for 1-D arrays.

a) Linear arrays
b) Lists
c) Horizontal array
d) Vertical array

https://www.youtube.com/watch?v=01sAkU_NvOY

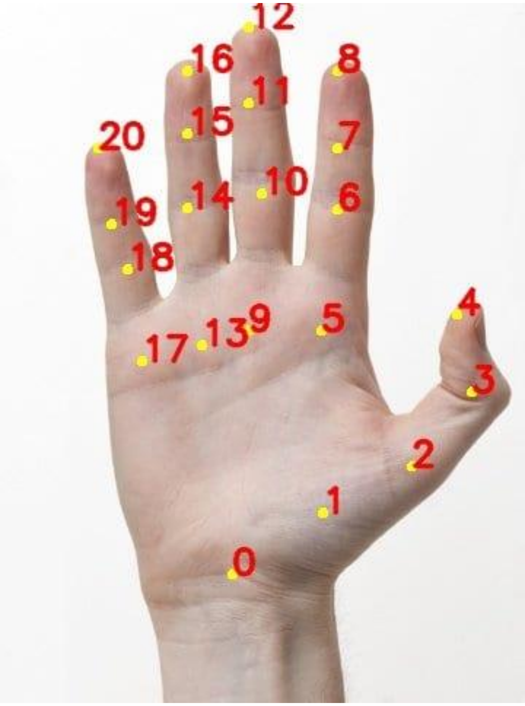
```
import cv2
import mediapipe as mp
import time
cap = cv2.VideoCapture(1)
mpHands = mp.solutions.hands
hands = mpHands.Hands()
mpDraw = mp.solutions.drawing_utils
while True:
    success, img = cap.read()
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    results = hands.process(imgRGB)
    if results.multi_hand_landmarks:
        for  handlms in results.multi_hand_landmarks:
            #21 points in handlms
            mpDraw.draw_landmarks(img, handlms,
mpHands.HAND_CONNECTIONS)

    cv2.imshow("Image", img)
    cv2.waitKey(1)
```

Assignment Project (20% and +2 maximum)



General Information

- Pick one of the following topics:
  - AI, DA, Blockchain or Software Engineering



AI (MediaPipe)

DA (World Cloud)

Example 1: Gesture Detection

Example 2: World Cloud Generation Software

Project Presentation

- Introduction to the project
- Use-Case diagram
- Sequence Diagram, Flowchart or Algorithm of one or two typical use-cases
- Implementation
- Results and Demo