when to use while loop.?

→ when we don't know, the number of times the loop has to run.

```
while(cond)
{
  // code.
  reintializatn
}.
```

Eg. Sum of digits
Reverse a Number
Palindrome

Math operatn in Java → Find number of digits →

```
int NOD = (int) Math.log10(n) + 1;
Println();
```

Program

```
static void main(string[] gs)
{
  int n = 21
  temp = n
  while (temp > 0)
  { sum = sum + temp % 10;
    prod = prod * temp / 10;
    rev = rev * 10 + temp / 10;
    temp = temp / 10;
  }
  println(sum)
  (prod)
  (rev)
```

for while v/s. do while

Eg. plain, ~~tick~~ train ticket booking.

First purchase, then enter → Evaluate

- Cond runs only when satisfies the conditions.

bus ticket booking.

first enter – then verify.

- Cond, where we know cond may be false, but want to run the loop atleast one –

ARRAYS.

→ int a[] = new int[]

→ Declaring >1 arrays- int [] a, b = new int[]; →
int ~~mar~~ a[], b[] = new int[]; →

Sorting

		Best	Worst	
Bubble	Approach: Comparison	$O(n^2)$	$O(n^2)$	
Selectn	Greedy	$O(n^2)$	$O(n^2)$	
Insertn	In-place Comparison	$O(n)$ [if sorted]	$O(n^2)$	
Quick	D&L	$O(n \log n)$	$O(n^2)$	pivot at 1st / last.
Merge	D&L	$O(n \log n)$	$O(n \log n)$	

BSTQM

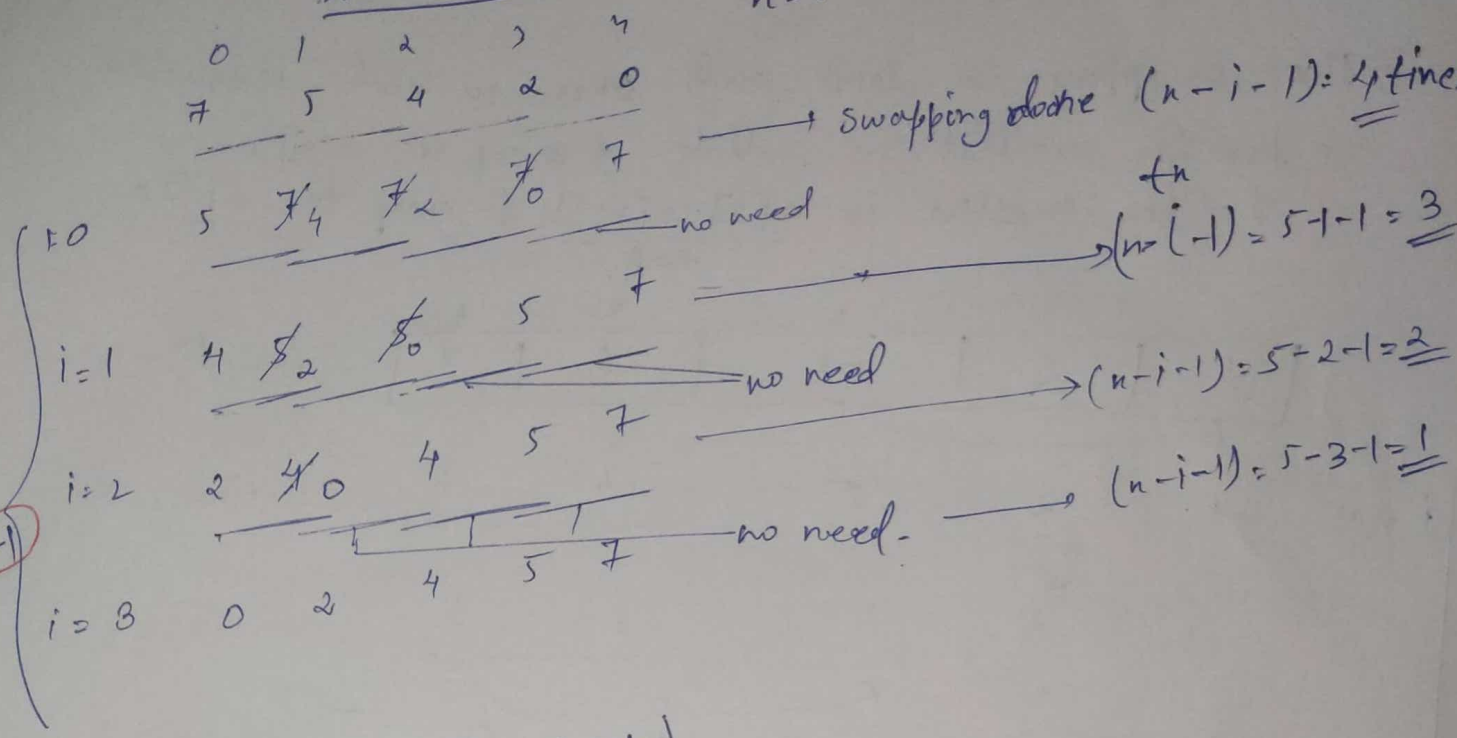
★ Bubble requires more comparison than selectn. (more t)

★ Insertn, Quick. — Similarities → use key / use pivot → Dissimilarities: if sorted $O(n)$; if sorted $O(n^2)$

Similarity in Quick & Merge.
- Both use D&C

Dissimilarity.
- exactly half division in Merge
- half division based on pivot - Quick.

Bubble Sort




```
for (i=0 ; i<n-1 ; i++)
{   boolean sorted = true;
    for (j=0 ; j<n-i-1 ; j++)
    {  if (a[j] > a[j+1])
       {  int temp = a[j+1];
          a[j+1] = a[j];
          a[j] = temp;   sorted = false;
       }
    }
    if (sorted) break;
}
for (int item : a)
    print (item + " ");
```

Optimization ①
Optimization ②

Suppose given array is already sorted, then how — or after 2, 3 elements rest are sorted — then how to skip further test.

if already sorted ⇒ no swap will occur, so the moment swap didn't occur for a traversal in j loop, break

11.

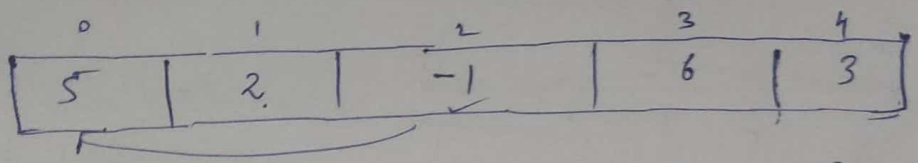
SELECTION SORT

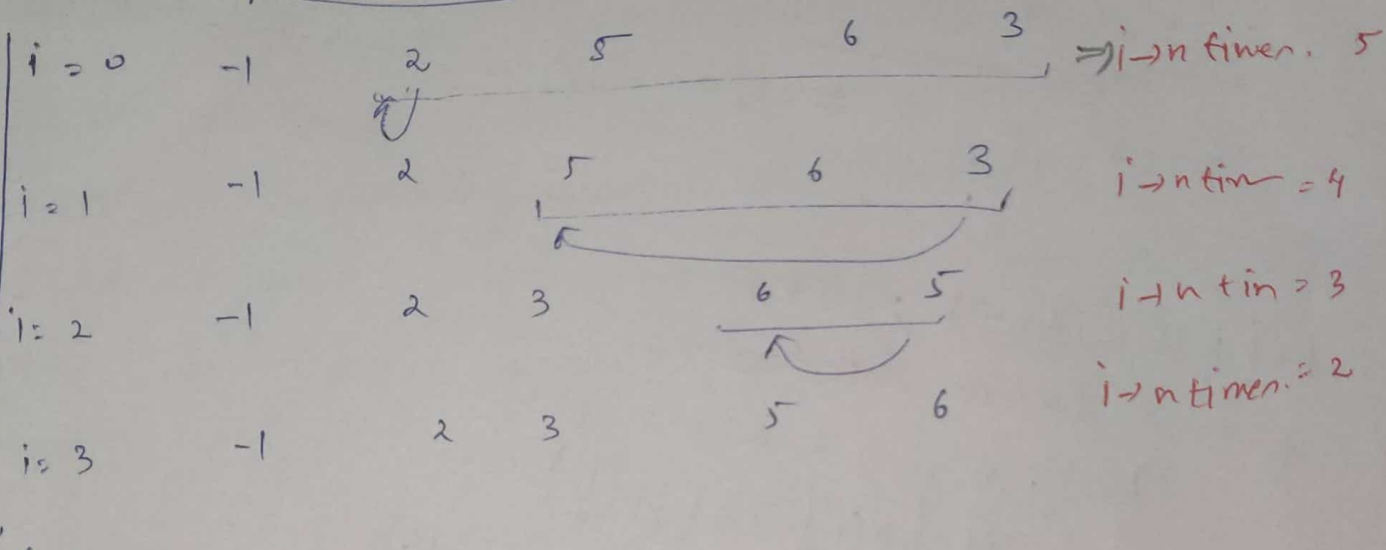
→ more optimized than Bubble

→ Here swapping is done only once in each iteration

- Find the smallest in whole & swap to a[0],
- Find the smallest in a[1]-a[n-1] & swap to a[1].

Rest.





1st loop

$i=0 \rightarrow i<n-1$

~~it~~ Inside i loop.

Eg $i=0 \rightarrow j=0$ 1 2 3 4

minInd = i

whenever a[j] < a[minInd] ⇒ { minInd = j }.

Now swap a[0] to a[minInd]

```
for( i=0;  i<n-1 ; i++)
{  int minInd = i
   for( j=0; j<n; j++)
      {  if (a[minInd] > a[j])
            { minInd = j },
      }.
   int temp = a[minInd];
      a[minInd] = a[i];
}        a[i] = a[minInd]; temp;
for (int e : a)
      print (*e + "_");
```