slido

What game are you looking forward to?

ⓘ Start presenting to display the poll results on this slide.

Recursion

Recursion

Recursion happens in code **when a method calls itself**.

This can happen when a method calls itself directly or when a method calls another method that then calls the original method.

Recursion: Exit Conditions

- Recursion is another kind of loop.
- Recursion however has some overhead which can cause a stack overflow exception if the method calls itself too many times.
- To prevent a stack overflow exception, you need to add an **exit condition** to the method so that it stops calling itself.

Recursion: Exit Conditions

- ALL recursive methods need an exit condition.
- An exit condition happens when a method completes or returns without calling itself again.

Recursion: Exit Conditions

- This recursive method will eventually cause a stack overflow exception because there is **no exit condition**.

```
unsigned long factorial(unsigned int N)
{
    return N * factorial(N - 1);
}
```

Recursion: Exit Conditions

- To fix this recursive method, we need to **add an exit condition** that will stop the method from calling itself.

```
unsigned long factorial(unsigned int N)
{
    if (N <= 1) return 1; //here's the exit condition!
    return N * factorial(N - 1);
}
```

Sorting

Why do we sort?

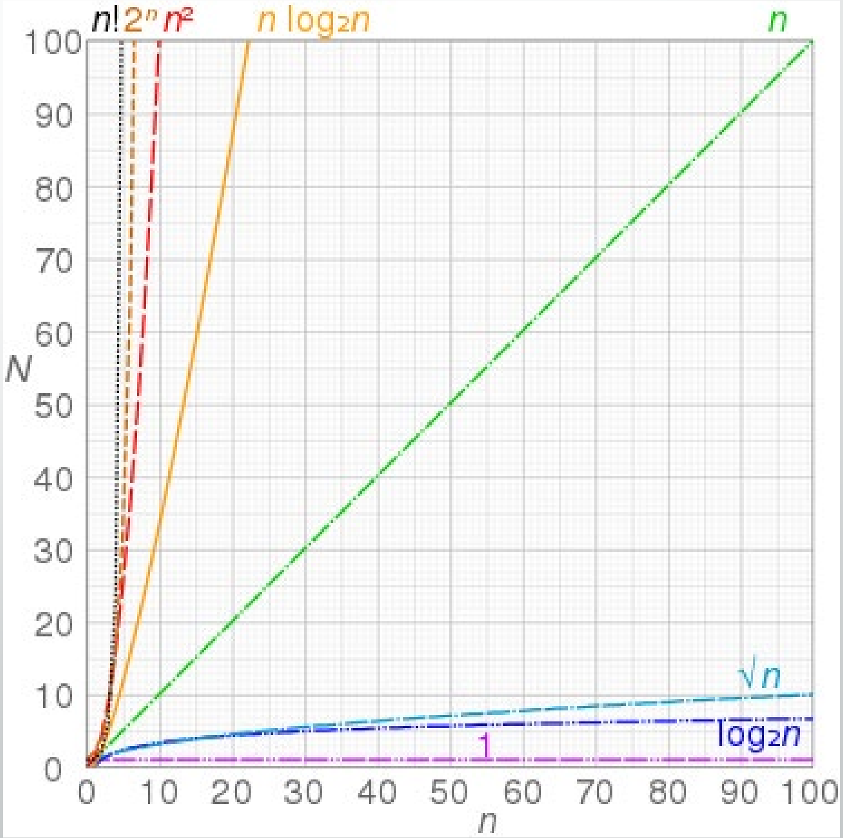
- Essential for searching and merging. The list must be sorted before it can be searched or merged efficiently.
- For UIs and user experience (ex: sort list by price low-high). Humans like to see lists arranged in ways that help them.
- For grouping and decision making (ex: Z-order)

Performance: Big-O notation

Best

Worst

O(1)	constant
O(log n)	logarithmic
O(n)	linear
O(n log n)	loglinear
$O(n^2)$	quadratic
$O(2^n)$	exponential
O(n!)	factorial



Sorting Performance

ALGORITHM	WORST	BEST	AVG
Heap Sort	O(n log n)	O(n log n)	O(n log n)
Quick Sort	$O(n^2)$	O(n log n)	O(n log n)
Merge Sort	O(n log n)	O(n log n)	O(n log n)
Insertion Sort	$O(n^2)$	O(n)	$O(n^2)$
Bubble Sort	$O(n^2)$ comparisons, $O(n^2)$ swaps	O(n) comps, O(1) swaps	$O(n^2)$ comparisons, $O(n^2)$ swaps

Bubble Sort

Bubble Sort Algorithm

- The Algorithm
  1. Compare each pair of adjacent elements from the beginning of the array
     - If they are in reversed order, swap them
  2. If at least one swap was made, repeat step 1

Bubble Sort Pseudocode

```
procedure bubbleSort(A : list of sortable items)
  n := length(A)
  repeat  //this is the start of a loop
    swapped := false
    for i := 1 to n - 1 inclusive do
      if A[i - 1] > A[i] then
        swap(A[i - 1], A[i]) //you'll need swap code here
        swapped = true
      end if
    end for
    n := n - 1
  while we swapped something //this is the end of the loop
end procedure
```

Swap Logic

Swap logic

- How to swap 2 items (A and B) in a vector
  - Copy A item to a temp variable.
  - Overwrite A item in the vector with the B item.
  - Overwrite the B item in the vector with the temp variable.

EXAMPLE: if vec holds ints…

```
int temp = vec[i – 1];
vec[i – 1] = vec[i];
vec[i] = temp;
```

Swap logic

- How to swap 2 items (A and B) in a vector
  - Use the std::swap method.
  - Use #include <string_view>

```
std::swap(vec[i - 1], vec[i]);
```

Comparing Strings

Comparing Strings

- When comparing strings for the purpose of sorting them, we need more information than whether the two strings are equal.
- We need to know “does this string come before the other string?”

Comparing Strings

- There are many ways to compare strings. We'll consider 2 approaches:
  - Using strcmp functions
  - Using std::string::compare
- Both approaches return an int value telling you how the strings compare.
  - So, comparing string s1 to string s2…
  - < 0 means s1 is LESS THAN (comes before) s2
  - 0 means s1 is EQUAL TO s2
  - > 0 means s1 is GREATER THAN (comes after) s2

Comparing Strings

- These versions of the strcmp function are case insensitive, meaning that it ignores the casing of the strings when comparing. Bob is equal to bob. _stricmp, _wcsicmp, _mbsicmp, _stricmp_l, _wcsicmp_l, _mbsicmp_l | Microsoft Learn

```
std::string s1 = "Batman", s2 = "Aquaman";
int compResult = _stricmp(s1.c_str(), s2.c_str());
```

** since _stricmp requires a pointer to a char array, you must call c_str() **

Comparing Strings

- std::string::compare
- Performs a case-sensitive comparison

```
std::string s1 = "Batman", s2 = "Aquaman";
int compareResult = s1.compare(s2);
```

If you want to do a case-insensitive comparison, one way would be to make both strings all uppercase or lowercase. (see next slide on how)

Comparing Strings

- Making a std::string uppercase.
- There are quite a few ways to make a string uppercase. Here is one way:
- `for (auto& c : str) c = toupper(c);`

- NOTE: this will modify the string. So if you want to keep the original string as-is, you’ll need to make a copy of the string and modify the copy.

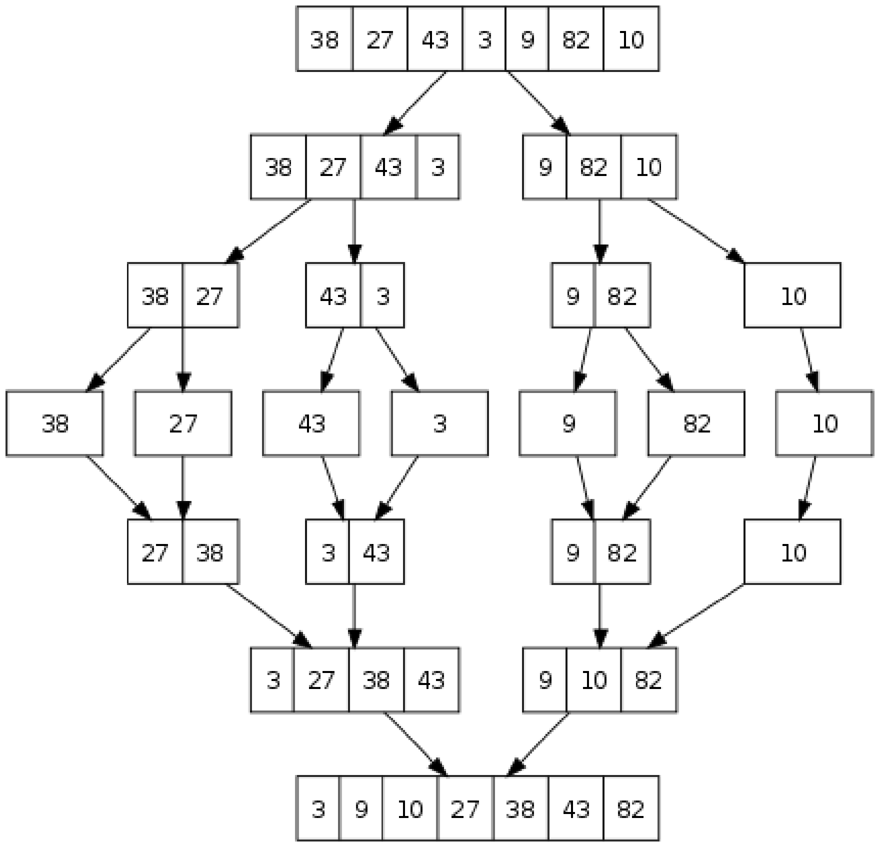
Merge Sort

Merge Sort Algorithm

- The Algorithm
  1. Divide the list into n sublists that are all 1 element long.
  2. Merge the sublists to make new ***sorted*** lists.
  3. Continue merging until only 1 list remains.

Merge Sort Algorithm

- https://en.wikipedia.org/wiki/File:Merge_sort_algorithm_diagram.svg



```
function merge_sort(list m) is
    // exit condition. A list of zero or one elements is sorted, by definition.
    if length of m ≤ 1 then
        return m

    // Recursive case. First, divide the list into sublists
    // consisting of the first half and second half of the list. This assumes lists start at index 0.

    var left := empty list
    var right := empty list

    for i = 0 to length(m) do
        if i < (length of m)/2 then
            add m[i] to left
        else
            add m[i] to right

    // Recursively sort both sublists.
    left := merge_sort(left)
    right := merge_sort(right)

    // Then merge the now-sorted sublists.
    return merge(left, right)
```

```
function merge(left, right) is
    var result := empty list

    while left is not empty and right is not empty do

        if first(left) ≤ first(right) then
            add first(left) to result
            remove first from left
        else
            add first(right) to result
            remove first from right

    // Either left or right may have elements left; consume them.
    // (Only one of the following loops will actually be entered.)
    while left is not empty do
        add first(left) to result
        remove first from left


    while right is not empty do
        add first(right) to result
        remove first from right

    return result
```

ALGORITHM	WORST	BEST	AVG
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick Sort	$O(n^2)$	$O(n \log n)$	$O(n \log n)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Insertion Sort	$O(n^2)$	$O(n)$	$O(n^2)$
Bubble Sort	$O(n^2)$ comparisons, $O(n^2)$ swaps	$O(n)$ comps, $O(1)$ swaps	$O(n^2)$ comparisons, $O(n^2)$ swaps

If you have 7 chairs and 10 children, what do you do?

slido

Recursion happens when...

ⓘ Start presenting to display the poll results on this slide.

slido

What do all recursive methods need?

ⓘ Start presenting to display the poll results on this slide.

slido

What happens when a recursive method calls itself too many times?

ⓘ Start presenting to display the poll results on this slide.

slido

What will be returned?

```
std::string s1 = "Batman", s2 = "Batmen";
int compResult = _stricmp(s1.c_str(), s2.c_str());
```

ⓘ Start presenting to display the poll results on this slide.