Overview of Recursion: Takeaways ⤴

Syntax

- Implementing recursion:

```python
example_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
def recursive_summation(values):
    if len(values) <= 1:
        return values[0]
    return values[0] + recursive_summation(values[1:])
recursive_summation(example_list)
```

- Implementing divide and conquer:

```python
f = open('random_integers.txt', 'r')
random_integers = [int(line) for line in f.readlines()]
def summation(values):
    if len(values) == 0:
        return 0
    if len(values) == 1:
        return values[0]
    midpoint = len(values) // 2
    return summation(values[:midpoint]) + summation(values[midpoint:])
divide_and_conquer_sum = summation(random_integers)
```

- Implementing merge sort:

```python
def merge(left_list, right_list):
sorted = []
while left_list and right_list:
    if left_list[0] < right_list[0]:
        sorted.append(left_list.pop(0))
    else:
        sorted.append(right_list.pop(0))
sorted += left_list
sorted += right_list
return sorted
```

Concepts

- Recursion is commonly defined as a function that calls itself.
- A terminating case makes sure recursion from continuing forever. The terminating case is also known as the base case.
- The base case is necessary to ensure your program doesn't run out of memory.
- A call stack is a stack data structure that stores information about the active subroutines of a computer program.
- A stack overflow occurs if the call stack pointer exceeds the stack bound. Stack overflow is a common cause of infinite recursion.
- Divide and conquer involves splitting the problem into a set of smaller sub problems that are easier. After reaching the easier terminal case, the values that will solve the general problem at hand gets returned.
- The goal of the merge sort algorithm is to first divide up an unsorted list into a bunch of smaller sorted lists and then merge them all together to create a sorted list.
- The time complexity for merge sort is $O(n\log_2 n)$.

Resources

- [Recursion](#)
- [Merge Sort Algorithm](#)