CP2410 Pracital 03

Sihan Chen, jcu ID: 14187662

> Q1. Describe a recursive algorithm for finding the maximum element in a sequence, S, of n elements. What is your running time and space usage?

```
def find_maximum(arr, n):
  if n == 1:
    return arr[0]
  else:
    return max(arr[n - 1], find_maximum(arr, n - 1))
```

Answer:

The code above will run recursively to find out maximum value.

Time complexity:

The code will run till n(the length of array) hits 1, in total it will execute n times, which gives it time complexity of O(n).

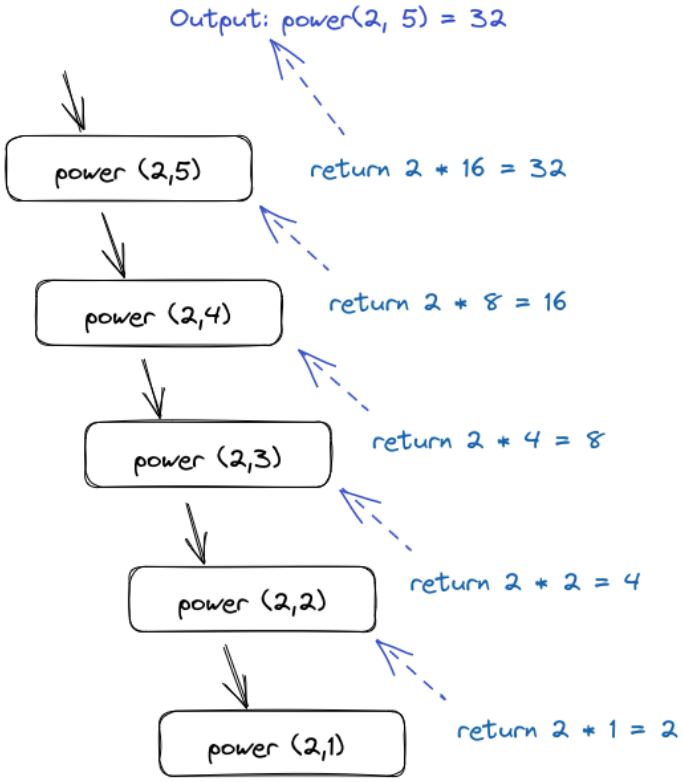
Space complexity:

The code run one time for base case, and n times for recursion, so the total space complexity is O(n+1)

> Q2. Draw the recursion trace for the computation of power(2,5), using the traditional function implemented below

```
def power(x, n):
  """ Compute the value x**n for integer n."""
  if n == 0:
    return 1
  else:
    return x * power(x, n - 1)
```

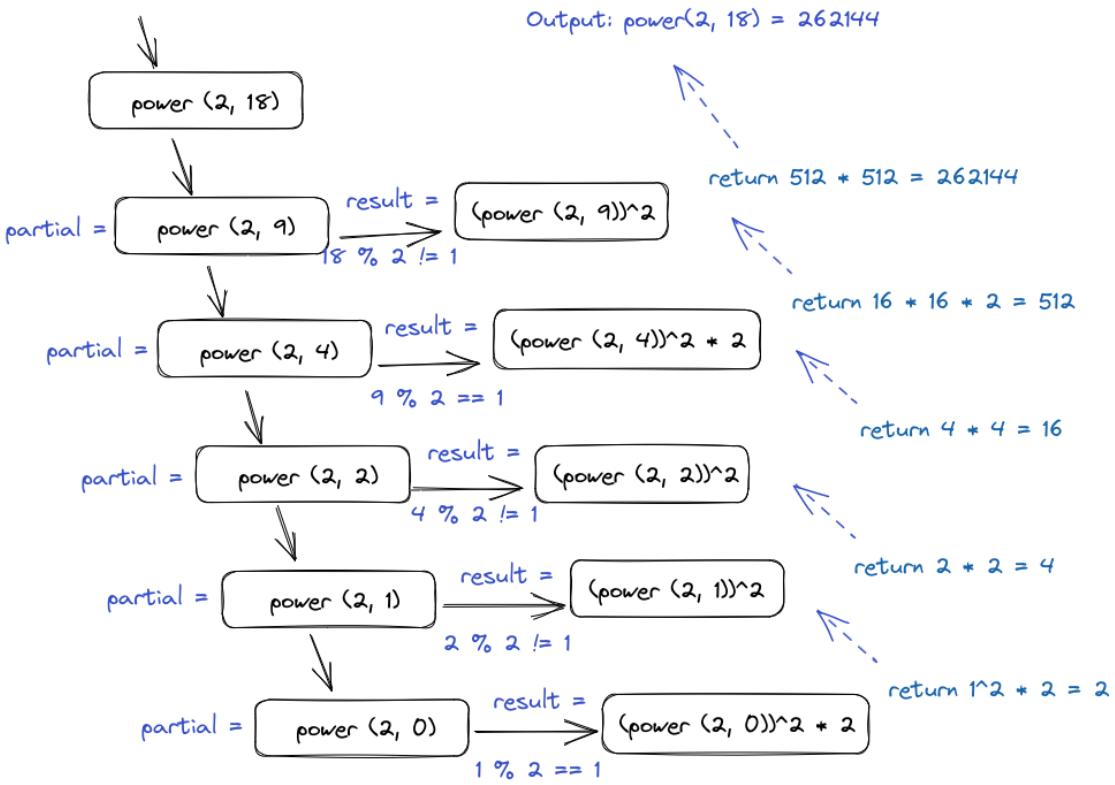
Answer:



> Q3. Draw the recursion trace for the computation of power(2,18), using the repeated squaring algorithm, as implemented below:

```
def power(x, n):
  """ Compute the value x**n for integer n."""
  if n == 0:
    return 1
  else:
    partial = power(x, n // 2)
    result = partial * partial
    if n % 2 == 1:
      result *= x
    return result
```

Answer:



> Q4. Give a recursive algorithm to compute the product of two positive integers, m and n, using only addition and subtraction

Answer:

```
def get_product(m, n):
  if n == 1:
    return m
  else:
    return m + get_product(m, n - 1)
```

The above code will calculate products of m and n. The base case is n equal to 1, just return m(m*1). If n is not 1, code run recursively till n eventually decrease to 1. Each time adding m to result. The code will run `len(n)` times, so the final result will be m * n.

> Q5. Modify ch05/experiment_list_append.py to investigate the time taken by append operations for DynamicArray (ch05/dynamic_array.py).

Answer:

Modification: import DynamicArray module, and for `data` object, initiate dynamic array instead of using list.

```
14 import sys
13 from time import time
12 from dynamic_array import DynamicArray
11
10 try:
 9     maxN = int(sys.argv[1])
 8 except:
 7     maxN = 10000000
 6
 5 def compute_average(n):
 4   """Perform n appends to an empty list and return average time elapsed."""
 3   data = DynamicArray()
 2   start = time()                  # record the start time (in seconds)
 1   for k in range(n):
36     data.append(None)
 1   end = time()                    # record the end time (in seconds)
 2   return (end - start) / n        # compute average per operation
 3
 4 n = 10
 5 while n <= maxN:
 6   print('Average of {0:.3f} for n {1}'.format(compute_average(n)*1000000, n))
 7   n *= 10
~
~
~
NORMAL  experiment_dynamic_array_append.py                          unix | u
```

Running result comparison:

```
❯ python experiment_list_append.py
Average of 0.596 for n 10
Average of 0.160 for n 100
Average of 0.059 for n 1000
Average of 0.058 for n 10000
Average of 0.057 for n 100000
Average of 0.046 for n 1000000
Average of 0.040 for n 10000000
❯ python experiment_dynamic_array_append.py
Average of 8.297 for n 10
Average of 0.770 for n 100
Average of 0.306 for n 1000
Average of 0.335 for n 10000
Average of 0.366 for n 100000
Average of 0.293 for n 1000000
Average of 0.344 for n 10000000
```

> Q6. Create a modified version of DynamicArray (ch05/dyanmic_array.py) that takes a parameter, resize_factor, which it uses to determine the new size (rather than doubling in the original code - self._resize(2 * self._capacity)). Using different values of resize_factor, examine if and how the average time to append changes.

Answer:

Modification: create a new class called DynamicArrayWithResize, which inherit from DynamicArray and takes in a resize factor value, override append method to multiply array size by resize factor.

```
import sys
from time import time
from dynamic_array import DynamicArray

try:
    maxN = int(sys.argv[1])
except:
    maxN = 10000000

class DynamicArrayWithResize(DynamicArray):
  def __init__(self, resize_factor):
    super().__init__()
    self._resize_factor = resize_factor

  def append(self, ele):
    # Not enough space in dynamic array
    if self._n == self._capacity:
      # add one in case resize_factor is set to 1
      self._resize(self._resize_factor * self._capacity + 1)
    self._A[self._n] = ele
    self._n += 1

def compute_average(n, i):
  """Perform n appends to an empty list and return average time elapsed."""
  data = DynamicArrayWithResize(i)
  start = time()                  # record the start time (in seconds)
  for k in range(n):
    data.append(None)
  end = time()                    # record the end time (in seconds)
  return (end - start) / n        # compute average per operation

n = 10
while n <= maxN:
  for i in range(2,5):
    print(f"Resize factor: {i}")
    print('Average of {0:.3f} for n {1}'.format(compute_average(n, i)*1000000, n))
  n *= 10
```

experiment_dynamic_array_append.py

Running program

```
python experiment_dynamic_array_append.py >> q6.txt
```

Here is the result:

Resize factor: 2 Average of 7.415 for n 10

Resize factor: 3 Average of 3.910 for n 10

Resize factor: 4 Average of 3.576 for n 10

Resize factor: 2 Average of 0.811 for n 100

Resize factor: 3 Average of 0.601 for n 100

Resize factor: 4 Average of 0.679 for n 100

Resize factor: 2 Average of 0.332 for n 1000

Resize factor: 3 Average of 0.271 for n 1000

Resize factor: 4 Average of 0.255 for n 1000

Resize factor: 2 Average of 0.396 for n 10000

Resize factor: 3 Average of 0.340 for n 10000

Resize factor: 4 Average of 0.296 for n 10000

Resize factor: 2 Average of 0.325 for n 100000

Resize factor: 3 Average of 0.315 for n 100000

Resize factor: 4 Average of 0.304 for n 100000

Resize factor: 2 Average of 0.279 for n 1000000

Resize factor: 3 Average of 0.284 for n 1000000

Resize factor: 4 Average of 0.206 for n 1000000

Resize factor: 2 Average of 0.345 for n 10000000

Resize factor: 3 Average of 0.282 for n 10000000

Resize factor: 4 Average of 0.242 for n 10000000