Tecnológico de Monterrey

Tecnológico de Monterrey - Campus Monterrey
School of Engineering and Sciences
Engineering in Computational Technologies
Analysis and Design of Advanced Algorithms

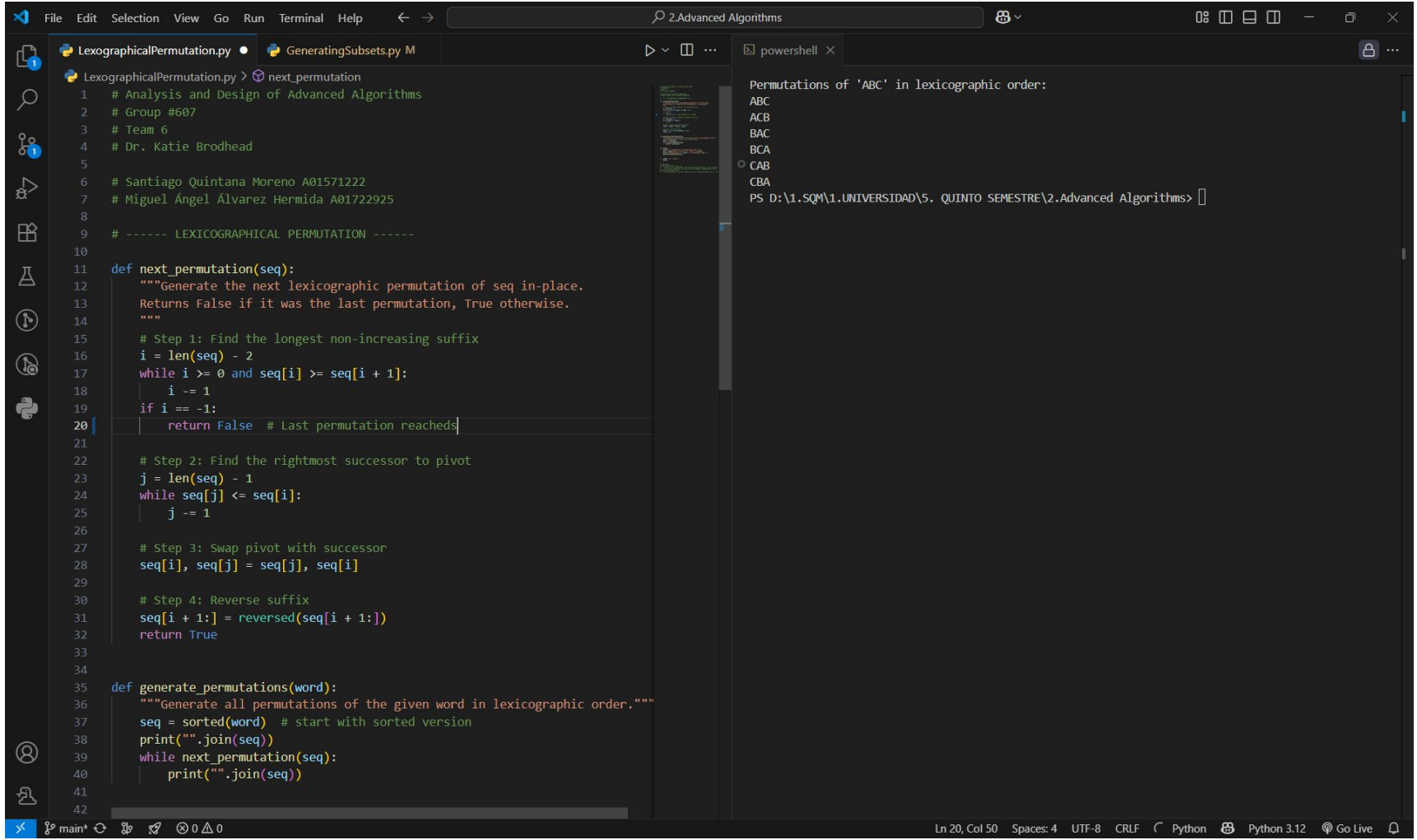
Homework 1:
Lexographical Order and Generating Subsets

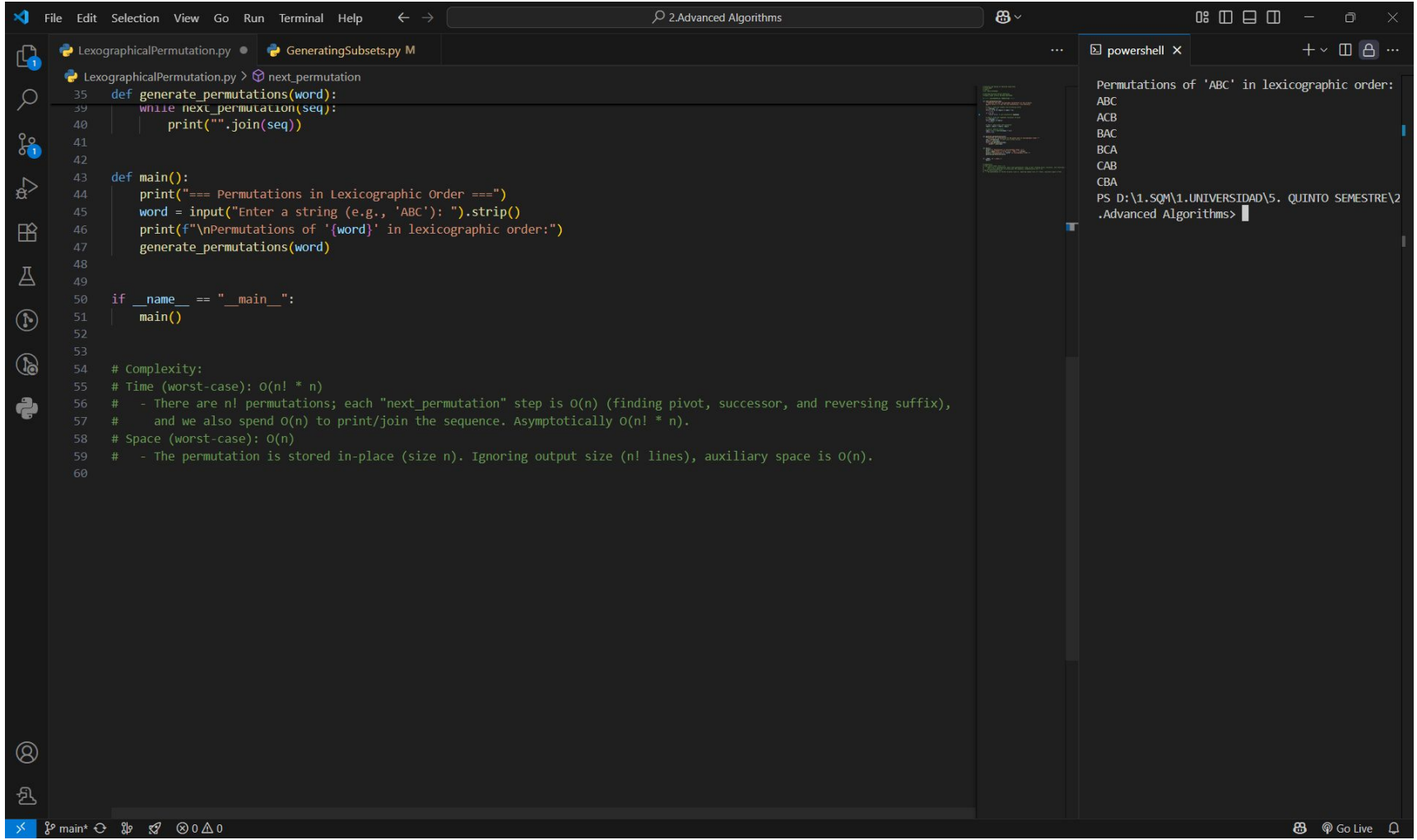
Group: 607
Team #6
Dr. Katie Brodhead

Santiago Quintana Moreno A01571222
Miguel Ángel Álvarez Hermida a01722925

Lexographical Permutations

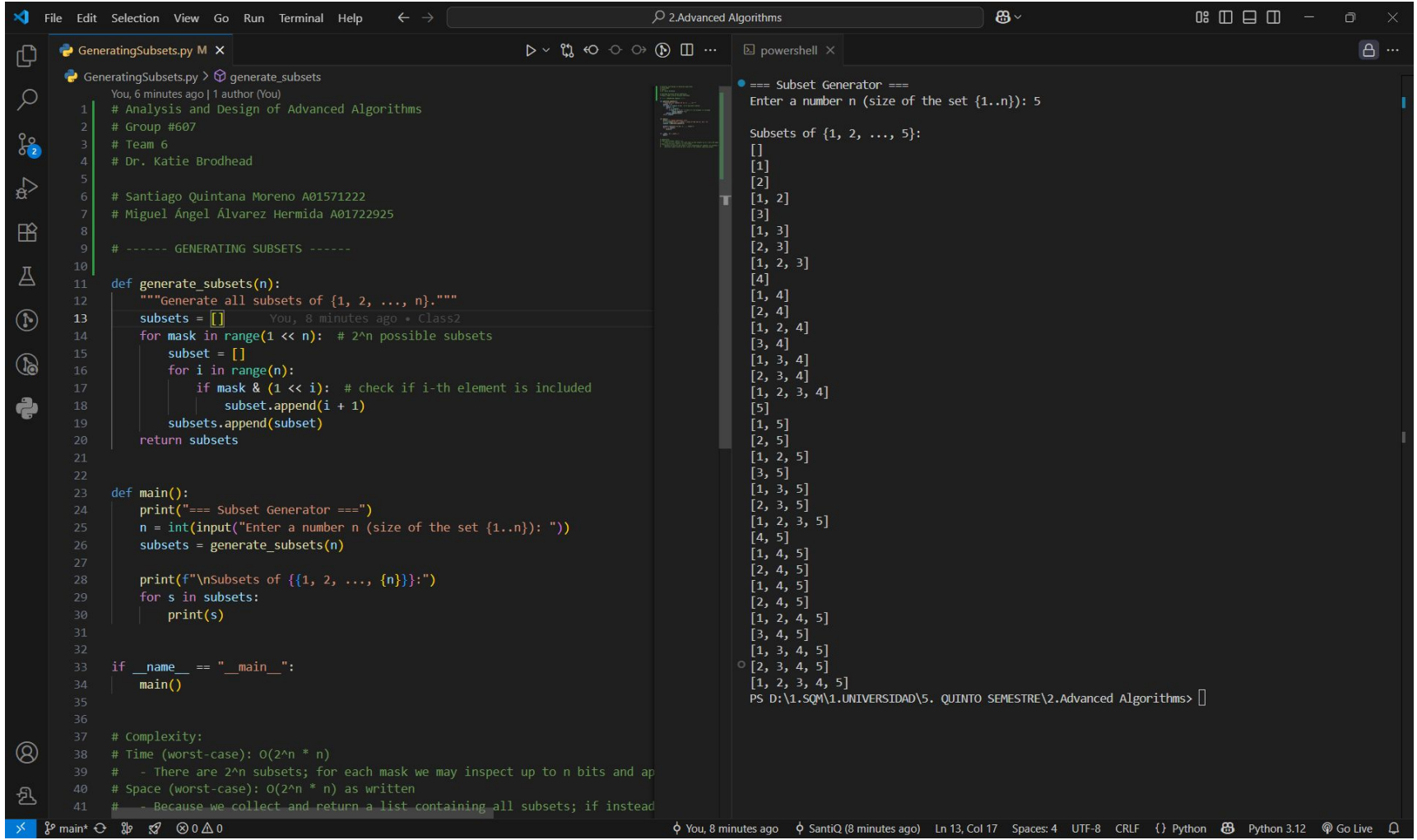
https://colab.research.google.com/drive/1-f1yMszXAh5ZhdmQyJ1WWuGqpcHYYcTe?usp=sharing

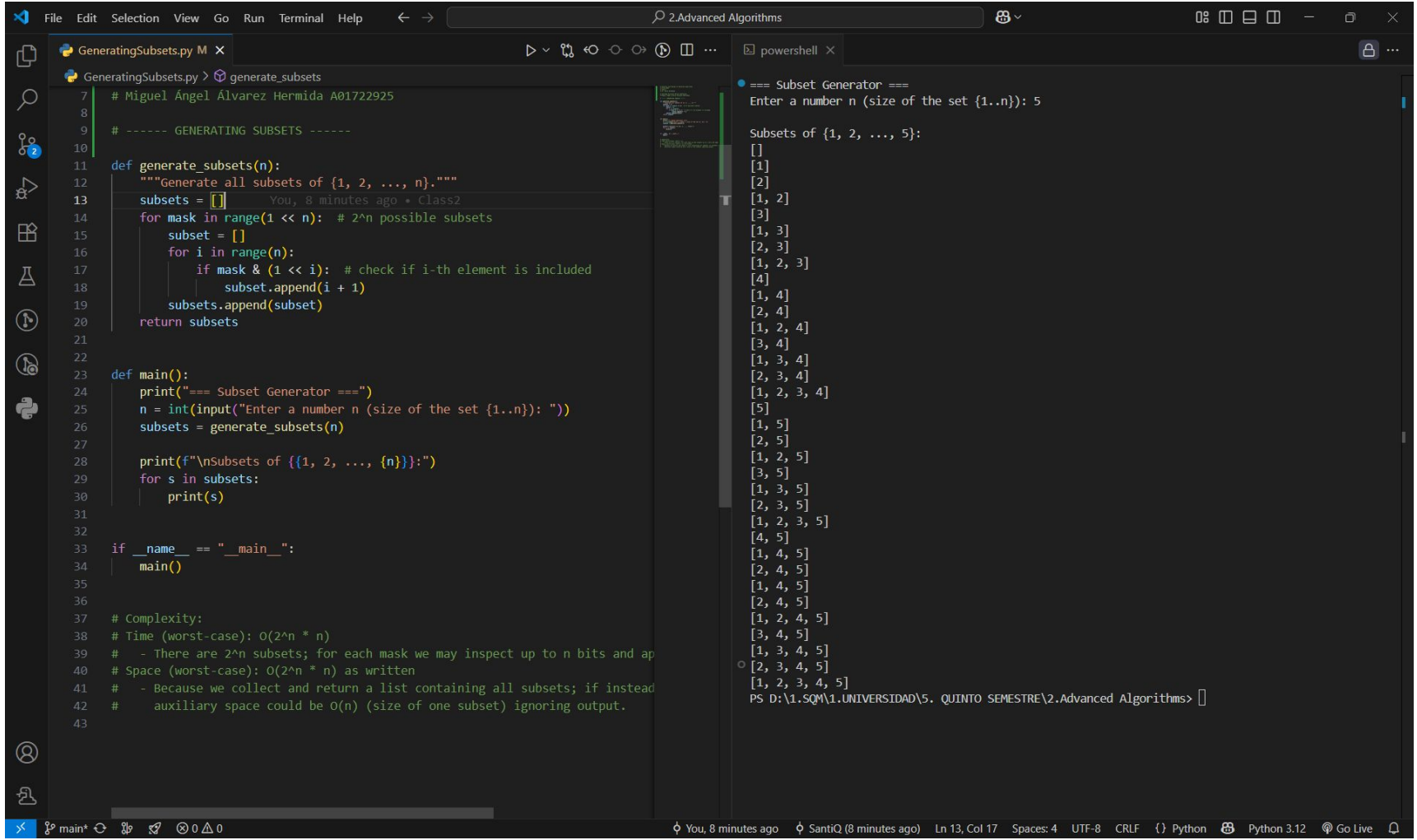

File Edit Selection View Go Run Terminal Help
2.Advanced Algorithms
LexographicalPermutation.py
GeneratingSubsets.py M
LexographicalPermutation.py > next_permutation
# Analysis and Design of Advanced Algorithms
# Group #607
# Team 6
# Dr. Katie Brodhead
# Santiago Quintana Moreno A01571222
# Miguel Ángel Álvarez Hermida A01722925
# ------ LEXICOGRAPHICAL PERMUTATION ------
def next_permutation(seq):
"""Generate the next lexicographic permutation of seq in-place.
Returns False if it was the last permutation, True otherwise.
"""
# Step 1: Find the longest non-increasing suffix
i = len(seq) - 2
while i >= 0 and seq[i] >= seq[i + 1]:
i -= 1
if i == -1:
return False # Last permutation reacheds
# Step 2: Find the rightmost successor to pivot
j = len(seq) - 1
while seq[j] <= seq[i]:
j -= 1
Step 3: Swap pivot with successor
seq[i], seq[j] = seq[j], seq[i]
Step 4: Reverse suffix
seq[i + 1:] = reversed(seq[i + 1:])
return True
def generate_permutations(word):
"""Generate all permutations of the given word in lexicographic order."""
seq = sorted(word) # start with sorted version
print("".join(seq))
while next_permutation(seq):
print("".join(seq))
powershell
Permutations of 'ABC' in lexicographic order:
ABC
ACB
BAC
BCA
CAB
CBA
PS D:\1.SQM\1.UNIVERSIDAD\5. QUINTO SEMESTRE\2.Advanced Algorithms>
main*
0 0
Ln 20, Col 50
Spaces: 4
UTF-8
CRLF
Python
Python 3.12
Go Live


File Edit Selection View Go Run Terminal Help
2.Advanced Algorithms
LexographicalPermutation.py
GeneratingSubsets.py M
LexographicalPermutation.py > next_permutation
def generate_permutations(word):
while next_permutation(seq):
print("".join(seq))
def main():
print("=== Permutations in Lexicographic Order ===")
word = input("Enter a string (e.g., 'ABC'): ").strip()
print(f"\nPermutations of '{word}' in lexicographic order:")
generate_permutations(word)
if __name__ == "__main__":
main()
# Complexity:
# Time (worst-case): O(n! * n)
# - There are n! permutations; each "next_permutation" step is O(n) (finding pivot, successor, and reversing suffix),
# and we also spend O(n) to print/join the sequence. Asymptotically O(n! * n).
# Space (worst-case): O(n)
# - The permutation is stored in-place (size n). Ignoring output size (n! lines), auxiliary space is O(n).
powershell
Permutations of 'ABC' in lexicographic order:
ABC
ACB
BAC
BCA
CAB
CBA
PS D:\1.SQM\1.UNIVERSIDAD\5. QUINTO SEMESTRE\2
.Advanced Algorithms>
main*
0 0
Go Live

Generating Subsets

https://colab.research.google.com/drive/1ySdGvl6d2H2NzNC7mrPUTSacu104b9vV?usp=sharing


File Edit Selection View Go Run Terminal Help
2.Advanced Algorithms
GeneratingSubsets.py M
GeneratingSubsets.py > generate_subsets
You, 6 minutes ago | 1 author (You)
# Analysis and Design of Advanced Algorithms
# Group #607
# Team 6
# Dr. Katie Brodhead
# Santiago Quintana Moreno A01571222
# Miguel Ángel Álvarez Hermida A01722925
# ------ GENERATING SUBSETS ------
def generate_subsets(n):
"""Generate all subsets of {1, 2, ..., n}."""
subsets = []
You, 8 minutes ago • Class2
for mask in range(1 << n): # 2^n possible subsets
subset = []
for i in range(n):
if mask & (1 << i): # check if i-th element is included
subset.append(i + 1)
subsets.append(subset)
return subsets
def main():
print("=== Subset Generator ===")
n = int(input("Enter a number n (size of the set {1..n}): "))
subsets = generate_subsets(n)
print(f"\nSubsets of {{1, 2, ..., {n}}}:")
for s in subsets:
print(s)
if __name__ == "__main__":
main()
# Complexity:
# Time (worst-case): O(2^n * n)
# - There are 2^n subsets; for each mask we may inspect up to n bits and ap
# Space (worst-case): O(2^n * n) as written
# - Because we collect and return a list containing all subsets; if instead
powershell
=== Subset Generator ===
Enter a number n (size of the set {1..n}): 5
Subsets of {1, 2, ..., 5}:
[]
[1]
[2]
[1, 2]
[3]
[1, 3]
[2, 3]
[1, 2, 3]
[4]
[1, 4]
[2, 4]
[1, 2, 4]
[3, 4]
[1, 3, 4]
[2, 3, 4]
[1, 2, 3, 4]
[5]
[1, 5]
[2, 5]
[1, 2, 5]
[3, 5]
[1, 3, 5]
[2, 3, 5]
[1, 2, 3, 5]
[4, 5]
[1, 4, 5]
[2, 4, 5]
[1, 4, 5]
[2, 4, 5]
[1, 2, 4, 5]
[3, 4, 5]
[1, 3, 4, 5]
[2, 3, 4, 5]
[1, 2, 3, 4, 5]
PS D:\1.SQM\1.UNIVERSIDAD\5. QUINTO SEMESTRE\2.Advanced Algorithms>
main*
0 0
You, 8 minutes ago
SantiQ (8 minutes ago)
Ln 13, Col 17
Spaces: 4
UTF-8
CRLF
Python
Python 3.12
Go Live


File Edit Selection View Go Run Terminal Help
2.Advanced Algorithms
GeneratingSubsets.py M
GeneratingSubsets.py > generate_subsets
# Miguel Ángel Álvarez Hermida A01722925
# ------ GENERATING SUBSETS ------
def generate_subsets(n):
"""Generate all subsets of {1, 2, ..., n}."""
subsets = []
You, 8 minutes ago • Class2
for mask in range(1 << n): # 2^n possible subsets
subset = []
for i in range(n):
if mask & (1 << i): # check if i-th element is included
subset.append(i + 1)
subsets.append(subset)
return subsets
def main():
print("=== Subset Generator ===")
n = int(input("Enter a number n (size of the set {1..n}): "))
subsets = generate_subsets(n)
print(f"\nSubsets of {{1, 2, ..., {n}}}:")
for s in subsets:
print(s)
if __name__ == "__main__":
main()
# Complexity:
# Time (worst-case): O(2^n * n)
# - There are 2^n subsets; for each mask we may inspect up to n bits and ap
# Space (worst-case): O(2^n * n) as written
# - Because we collect and return a list containing all subsets; if instead
# auxiliary space could be O(n) (size of one subset) ignoring output.
powershell
=== Subset Generator ===
Enter a number n (size of the set {1..n}): 5
Subsets of {1, 2, ..., 5}:
[]
[1]
[2]
[1, 2]
[3]
[1, 3]
[2, 3]
[1, 2, 3]
[4]
[1, 4]
[2, 4]
[1, 2, 4]
[3, 4]
[1, 3, 4]
[2, 3, 4]
[1, 2, 3, 4]
[5]
[1, 5]
[2, 5]
[1, 2, 5]
[3, 5]
[1, 3, 5]
[2, 3, 5]
[1, 2, 3, 5]
[4, 5]
[1, 4, 5]
[2, 4, 5]
[1, 4, 5]
[2, 4, 5]
[1, 2, 4, 5]
[3, 4, 5]
[1, 3, 4, 5]
[2, 3, 4, 5]
[1, 2, 3, 4, 5]
PS D:\1.SQM\1.UNIVERSIDAD\5. QUINTO SEMESTRE\2.Advanced Algorithms>
main*
You, 8 minutes ago
SantiQ (8 minutes ago)
Ln 13, Col 17
Spaces: 4
UTF-8
CRLF
Python
Python 3.12
Go Live