1. Factorial Calculation

```
#!/bin/bash
echo -n "Enter a number: "
read num
fact=1
for ((i=1; i<=num; i++)); do
  fact=$((fact * i))
done
echo "Factorial of $num is $fact"
```

2. Check if a Number is Prime

```
#!/bin/bash
echo -n "Enter a number: "
read num
is_prime=1
for ((i=2; i*i<=num; i++)); do
  if ((num % i == 0)); then
    is_prime=0
    break
  fi
done
((is_prime && num > 1)) && echo "$num is Prime" || echo "$num is Not Prime"
```

3. Fibonacci Series (N Terms)

```
#!/bin/bash
echo -n "Enter N: "
read n
a=0
b=1
echo "Fibonacci Series: "
for ((i=0; i<n; i++)); do
  echo -n "$a "
  fn=$((a + b))
```

```
  a=$b
  b=$fn
done
echo
```

4. Reverse a String

```
#!/bin/bash
echo -n "Enter a string: "
read str
rev=""
for ((i=${#str}-1; i>=0; i--)); do
  rev="$rev${str:i:1}"
done
echo "Reversed String: $rev"
```

5. Count Number of Files in a Directory

```
#!/bin/bash
echo -n "Enter directory path: "
read dir
count=$(ls -1 "$dir" | wc -l)
echo "Total files in $dir: $count"
```

6. Find the Largest of Three Numbers

```
#!/bin/bash
echo -n "Enter three numbers: "
read a b c
echo "Largest number is: $((a > b ? (a > c ? a : c) : (b > c ? b : c)))"
```

7. Sum of Digits of a Number

```
#!/bin/bash
echo -n "Enter a number: "
read num
sum=0
while [ $num -gt 0 ]; do
  sum=$((sum + num % 10))
```

```
  num=$((num / 10))
done
echo "Sum of digits: $sum"
```

8. Convert Celsius to Fahrenheit

```
#!/bin/bash
echo -n "Enter temperature in Celsius: "
read c
f=$(echo "scale=2; ($c * 9/5) + 32" | bc)
echo "$c°C = $f°F"
```

9. Check if a String is a Palindrome

```
#!/bin/bash
echo -n "Enter a string: "
read str
rev=$(echo "$str" | rev)
[[ "$str" == "$rev" ]] && echo "Palindrome" || echo "Not a Palindrome"
```

10. Display Even Numbers from 1 to N

```
#!/bin/bash
echo -n "Enter N: "
read n
echo "Even numbers from 1 to $n:"
for ((i=2; i<=n; i+=2)); do
  echo -n "$i "
done
echo
```

11. Find the GCD of Two Numbers

```
#!/bin/bash
gcd() {
```

```
  (( $2 == 0 )) && echo $1 || gcd $2 $(( $1 % $2 ))
}
echo -n "Enter two numbers: "
read a b
echo "GCD: $(gcd $a $b)"
```

12. Print a Pyramid Pattern

```
#!/bin/bash
echo -n "Enter the number of rows: "
read rows
for ((i=1; i<=rows; i++)); do
  for ((j=1; j<=rows-i; j++)); do echo -n " "; done
  for ((k=1; k<=2*i-1; k++)); do echo -n "*"; done
  echo
done
```

13. Check if a File Exists

```
#!/bin/bash
echo -n "Enter filename: "
read file
[[ -f "$file" ]] && echo "File exists" || echo "File not found"
```

14. Calculate Power of a Number (a^b)

```
#!/bin/bash
echo -n "Enter base and exponent: "
read base exp
result=1
for ((i=0; i<exp; i++)); do
  result=$((result * base))
done
echo "$base^$exp = $result"
```

15. Generate a Random Password

```
#!/bin/bash
echo "Random Password: $(tr -dc 'A-Za-z0-9@#$%&*' </dev/urandom | head -c 12)"
```

16. Find and Replace Text in Multiple Files

```
#!/bin/bash
echo -n "Enter directory: "
read dir
echo -n "Find: "
read find
echo -n "Replace with: "
read replace
grep -rl "$find" "$dir" | xargs sed -i "s/$find/$replace/g"
echo "Text replaced in all matching files!"
```

17. Show Battery Percentage (Linux Only)

```
#!/bin/bash
echo "Battery: $(cat /sys/class/power_supply/BAT0/capacity)%"
```

18. Convert All PNGs to JPGs in a Folder

```
#!/bin/bash
echo -n "Enter folder path: "
read folder
for img in "$folder"/*.png; do
  convert "$img" "${img%.png}.jpg"
done
echo "Conversion completed!"
```

20. Create a Backup of a Folder with Date & Time

```
#!/bin/bash
echo -n "Enter folder to backup: "
read folder
tar -czf "${folder}_backup_$(date +%Y%m%d_%H%M%S).tar.gz" "$folder"
```

```
echo "Backup created successfully!"
```

21. Get the Public IP Address

```
#!/bin/bash
echo "Your Public IP: $(curl -s ifconfig.me)"
```

22. Get Weather Report of Any City

```
#!/bin/bash
echo -n "Enter city: "
read city
curl -s "wttr.in/$city?format=3"
```

23. Countdown Timer (Self-Destruct Mode!)

```
#!/bin/bash
echo -n "Enter countdown time (in seconds): "
read sec
while [ $sec -gt 0 ]; do
  echo -ne "Self-destruct in $sec seconds...\r"
  sleep 1
  ((sec--))
done
echo -e "\nBOOM! 💥"
```

24. Monitor CPU Temperature (Linux Only)

```
#!/bin/bash
echo "CPU Temperature: $(sensors | grep 'Package id 0' | awk '{print $4}')"
```

25. Make Your Terminal Talk! (Linux/macOS Only)

```
#!/bin/bash
echo -n "What should I say? "
read text
```

```
espeak "$text"
```

26. Bubble Sort (Sort an Array)

```
#!/bin/bash
echo -n "Enter numbers (space-separated): "
read -a arr
n=${#arr[@]}

for ((i=0; i<n-1; i++)); do
  for ((j=0; j<n-i-1; j++)); do
    if (( arr[j] > arr[j+1] )); then
      temp=${arr[j]}
      arr[j]=${arr[j+1]}
      arr[j+1]=$temp
    fi
  done
done

echo "Sorted Array: ${arr[@]}"
```

27. Binary Search (Find an Element in a Sorted Array)

```
#!/bin/bash
echo -n "Enter sorted numbers (space-separated): "
read -a arr
echo -n "Enter number to search: "
read target

low=0
high=$((${#arr[@]} - 1))

while (( low <= high )); do
  mid=$(((low + high) / 2))
  if (( arr[mid] == target )); then
    echo "Found at index $mid"
    exit
  elif (( arr[mid] < target )); then
    low=$((mid + 1))
  else
    high=$((mid - 1))
  fi
done

echo "Not found!"
```

28. Greatest Common Divisor (GCD) using Euclidean Algorithm

```
#!/bin/bash
gcd() {
  (( $2 == 0 )) && echo $1 || gcd $2 $(( $1 % $2 ))
}
```

```
echo -n "Enter two numbers: "
read a b
echo "GCD: $(gcd $a $b)"
```

29. Tower of Hanoi (Recursive Algorithm)

```
#!/bin/bash
hanoi() {
  (( $1 == 1 )) && echo "Move disk 1 from $2 to $3" && return
  hanoi $(($1-1)) $2 $4 $3
  echo "Move disk $1 from $2 to $3"
  hanoi $(($1-1)) $4 $3 $2
}

echo -n "Enter number of disks: "
read n
hanoi $n A C B
```

30. Dijkstra's Algorithm (Find Shortest Path in a Graph)

```
#!/bin/bash
declare -A graph
graph[A,B]=4
graph[A,C]=1
graph[B,C]=2
graph[B,D]=5
graph[C,D]=8

echo "Graph Edges (Format: Node1,Node2=Weight):"
for key in "${!graph[@]}"; do echo "$key=${graph[$key]}"; done
```

31. Merge Sort (Divide and Conquer)

```
#!/bin/bash

merge_sort() {
  local arr=("$@")
  local n=${#arr[@]}
  (( n < 2 )) && echo "${arr[@]}" && return


  local mid=$(( n / 2 ))
  local left=("${arr[@]:0:mid}")
  local right=("${arr[@]:mid}")
```

```
  left=($(merge_sort "${left[@]}"))
  right=($(merge_sort "${right[@]}"))

  merge "${left[@]}" "${right[@]}"
}

merge() {
  local left=("$@")
  local right=("${left[@]:$(( (${#left[@]}) / 2 ))}")
  left=("${left[@]:0:$(( (${#left[@]}) / 2 ))}")

  local merged=()
  while [[ ${#left[@]} -gt 0 && ${#right[@]} -gt 0 ]]; do
    if (( left[0] <= right[0] )); then
      merged+=("${left[0]}")
      left=("${left[@]:1}")
    else
      merged+=("${right[0]}")
      right=("${right[@]:1}")
    fi
  done
  echo "${merged[@]}" "${left[@]}" "${right[@]}"
}

echo -n "Enter numbers (space-separated): "
read -a arr
echo "Sorted Array: $(merge_sort "${arr[@]}")"
```

32. Josephus Problem (Survivor in a Circle)

```
#!/bin/bash
```

```
josephus() {
  (( $1 == 1 )) && echo 0 || echo $(( ( $(josephus $(($1-1))) + $2 ) % $1 ))
}

echo -n "Enter number of people: "
read n
echo -n "Enter step count: "
read k
echo "Safe position: $(( $(josephus $n $k) + 1 ))"
```

33. Check if a Number is Armstrong

```
#!/bin/bash
echo -n "Enter a number: "
read num
sum=0
n=$num
len=${#num}

while (( n > 0 )); do
  digit=$(( n % 10 ))
  sum=$(( sum + digit ** len ))
  n=$(( n / 10 ))
done

(( sum == num )) && echo "$num is an Armstrong number" || echo "$num is not an Armstrong number"
```

34. Kadane’s Algorithm (Max Subarray Sum)

```
#!/bin/bash
echo -n "Enter numbers (space-separated): "
read -a arr
max_sum=${arr[0]}
```

```
current_sum=0

for num in "${arr[@]}"; do
  (( current_sum < 0 )) && current_sum=0
  current_sum=$(( current_sum + num ))
  (( current_sum > max_sum )) && max_sum=$current_sum
done

echo "Maximum Subarray Sum: $max_sum"
```

35. Checking if a String is a Palindrome

```
#!/bin/bash
echo -n "Enter a string: "
read str
rev=$(echo "$str" | rev)

[[ "$str" == "$rev" ]] && echo "Palindrome" || echo "Not a Palindrome"
```

36. Quick Sort (Divide and Conquer)

```
#!/bin/bash
quick_sort() {
  local arr=("$@")
  local n=${#arr[@]}
  (( n < 2 )) && echo "${arr[@]}" && return

  local pivot=${arr[0]}
  local left=() right=()

  for ((i=1; i<n; i++)); do
    (( arr[i] < pivot )) && left+=("${arr[i]}") || right+=("${arr[i]}")
  done
```

```
  echo "$(quick_sort "${left[@]}") $pivot $(quick_sort "${right[@]}")"
}

echo -n "Enter numbers (space-separated): "
read -a arr
echo "Sorted Array: $(quick_sort "${arr[@]}")"
```

37. Counting Sort (Non-Comparison Sorting)

```
#!/bin/bash
echo -n "Enter numbers (space-separated): "
read -a arr

max=0
for num in "${arr[@]}"; do
  (( num > max )) && max=$num
done

declare -a count
for num in "${arr[@]}"; do
  ((count[num]++))
done

sorted=()
for ((i=0; i<=max; i++)); do
  for ((j=0; j<count[i]; j++)); do
    sorted+=("$i")
  done
done

echo "Sorted Array: ${sorted[@]}"
```

38. N-Queens Problem (Backtracking)

```
#!/bin/bash
N=4
board=()

is_safe() {
  for ((i=0; i<$1; i++)); do
    (( board[i] == board[$1] || abs $((board[i] - board[$1])) == abs $((i - $1)) )) && return 1
  done
  return 0
}

abs() { echo $(( $1 < 0 ? -$1 : $1 )); }

solve_nqueens() {
  (( $1 == N )) && { echo "${board[@]}"; return; }
  for ((col=0; col<N; col++)); do
    board[$1]=$col
    is_safe $1 && solve_nqueens $(( $1 + 1 ))
  done
}

solve_nqueens 0
```

39. Reverse an Integer Without Using String Operations

```
#!/bin/bash
echo -n "Enter a number: "
read num
rev=0

while (( num > 0 )); do
```

```
  rem=$(( num % 10 ))
  rev=$(( rev * 10 + rem ))
  num=$(( num / 10 ))
done

echo "Reversed Number: $rev"
```

40. Pascal's Triangle (Combinatorial Mathematics)

```
#!/bin/bash
echo -n "Enter number of rows: "
read n

for ((i=0; i<n; i++)); do
  num=1
  for ((j=0; j<=i; j++)); do
    echo -n "$num "
    num=$(( num * (i - j) / (j + 1) ))
  done
  echo
done
```

41. Binary Search (Efficient Search Algorithm)

```
#!/bin/bash
binary_search() {
  local arr=("$@")
  local key=${arr[-1]}
  unset arr[-1]  # Remove the key from the array
  local left=0 right=$(( ${#arr[@]} - 1 ))

  while (( left <= right )); do
    mid=$(( (left + right) / 2 ))
    if (( arr[mid] == key )); then
```

```
      echo "Found at index $mid"
      return
    elif (( arr[mid] < key )); then
      left=$(( mid + 1 ))
    else
      right=$(( mid - 1 ))
    fi
  done
  echo "Not Found"
}

echo -n "Enter sorted numbers (space-separated): "
read -a arr
echo -n "Enter the number to search: "
read key
binary_search "${arr[@]}" "$key"
```

42. Greatest Common Divisor (GCD - Euclidean Algorithm)

```
#!/bin/bash
gcd() {
  (( $2 == 0 )) && echo "$1" || gcd "$2" "$(( $1 % $2 ))"
}

echo -n "Enter two numbers: "
read a b
echo "GCD: $(gcd $a $b)"
```

43. Sieve of Eratosthenes (Find Prime Numbers Efficiently)

```
#!/bin/bash
echo -n "Enter the upper limit: "
read n
```

```
primes=($(seq 2 $n))
for ((i=2; i*i<=n; i++)); do
  for ((j=i*i; j<=n; j+=i)); do
    primes[j]=""
  done
done

echo "Prime numbers up to $n: ${primes[@]}"
```

44. Dijkstra’s Algorithm (Shortest Path in a Graph)

```
#!/bin/bash
INF=99999
declare -A graph

read_graph() {
  echo -n "Enter number of vertices: "
  read n
  for ((i=0; i<n; i++)); do
    for ((j=0; j<n; j++)); do
      echo -n "Enter weight from $i to $j (INF=$INF): "
      read graph[$i,$j]
    done
  done
}

dijkstra() {
  local src=$1
  local dist=()
  local visited=()
```

```
  for ((i=0; i<n; i++)); do
    dist[i]=$INF
    visited[i]=0
  done
  dist[$src]=0

  for ((count=0; count<n-1; count++)); do
    min=$INF
    for ((v=0; v<n; v++)); do
      if (( !visited[v] && dist[v] < min )); then
        min=${dist[v]}
        u=$v
      fi
    done

    visited[u]=1
    for ((v=0; v<n; v++)); do
      if (( !visited[v] && graph[$u,$v] != INF && dist[u] + graph[$u,$v] < dist[v] )); then
        dist[v]=$(( dist[u] + graph[$u,$v] ))
      fi
    done
  done

  echo "Vertex  Distance from Source"
  for ((i=0; i<n; i++)); do
    echo "$i        ${dist[i]}"
  done
}

read_graph
```

```
echo -n "Enter source vertex: "
read src
dijkstra $src
```

45. Tower of Hanoi (Recursive Algorithm)

```
#!/bin/bash
hanoi() {
  (( $1 == 1 )) && echo "Move disk 1 from $2 to $3" && return
  hanoi $(( $1 - 1 )) $2 $4 $3
  echo "Move disk $1 from $2 to $3"
  hanoi $(( $1 - 1 )) $4 $3 $2
}

echo -n "Enter number of disks: "
read n
hanoi $n A C B
```

46. Knapsack Problem (Dynamic Programming)

```
#!/bin/bash
knapsack() {
  local -n weights=$1
  local -n values=$2
  local capacity=$3
  local n=${#weights[@]}
  declare -A dp

  for ((i=0; i<=n; i++)); do
    for ((w=0; w<=capacity; w++)); do
      if (( i == 0 || w == 0 )); then
        dp[$i,$w]=0
      elif (( weights[i-1] <= w )); then
        dp[$i,$w]=$(( values[i-1] + dp[i-1,w-weights[i-1]] > dp[i-1,w] ? values[i-1] + dp[i-1,w-
weights[i-1]] : dp[i-1,w] ))
```

```
      else
        dp[$i,$w]=${dp[i-1,w]}
      fi
    done
  done

  echo "Maximum value in Knapsack: ${dp[$n,$capacity]}"
}

weights=(2 3 4 5)
values=(3 4 5 6)
capacity=5
knapsack weights values $capacity
```

47. N-Queens Problem (Backtracking)

```
#!/bin/bash
n_queens() {
  local n=$1
  local -a board
  solve 0 $n board
}

is_safe() {
  local row=$1 col=$2
  local -n board=$3

  for ((i=0; i<row; i++)); do
    [[ ${board[i]} -eq $col || $(( board[i] - i )) -eq $(( col - row )) || $(( board[i] + i )) -eq $(( col + row )) ]] && return 1
  done
  return 0
}
```

```
solve() {
  local row=$1 n=$2
  local -n board=$3

  (( row == n )) && { echo "Solution: ${board[@]}"; return; }

  for ((col=0; col<n; col++)); do
    is_safe $row $col board && { board[row]=$col; solve $((row+1)) $n board; }
  done
}

echo -n "Enter board size (N): "
read n
n_queens $n
```

48. Rabin-Karp String Search (Pattern Matching)

```
#!/bin/bash
rabin_karp() {
  local text="$1" pattern="$2"
  local m=${#pattern} n=${#text} prime=101
  local hash_pat=0 hash_txt=0 h=1 i j

  for ((i=0; i<m-1; i++)); do
    h=$(( h * 256 % prime ))
  done

  for ((i=0; i<m; i++)); do
    hash_pat=$(( (256 * hash_pat + ${pattern:i:1}) % prime ))
    hash_txt=$(( (256 * hash_txt + ${text:i:1}) % prime ))
  done
```

```
  for ((i=0; i<=n-m; i++)); do
    [[ $hash_pat -eq $hash_txt && "${text:i:m}" == "$pattern" ]] && echo "Pattern found at index $i"
    [[ $i -lt $((n-m)) ]] && hash_txt=$(( (256 * (hash_txt - ${text:i:1} * h) + ${text:i+m:1}) % prime ))
  done
}

echo -n "Enter text: "
read text
echo -n "Enter pattern: "
read pattern
rabin_karp "$text" "$pattern"
```

49. KMP Algorithm (Optimized String Matching)

```
#!/bin/bash
kmp_search() {
  local text="$1" pattern="$2"
  local n=${#text} m=${#pattern}
  local -a lps

  compute_lps "$pattern" lps

  local i=0 j=0
  while (( i < n )); do
    if [[ ${text:i:1} == ${pattern:j:1} ]]; then
      (( j++, i++ ))
      (( j == m )) && { echo "Pattern found at index $((i - j))"; j=${lps[j-1]}; }
    else
      (( j > 0 )) && j=${lps[j-1]} || (( i++ ))
    fi
  done
```

```
}

compute_lps() {
  local pat="$1"
  local -n lps=$2
  local len=0 i=1
  lps[0]=0

  while (( i < ${#pat} )); do
    if [[ ${pat:i:1} == ${pat:len:1} ]]; then
      (( len++, lps[i]=len, i++ ))
    else
      (( len > 0 )) && len=${lps[len-1]} || (( lps[i]=0, i++ ))
    fi
  done
}

echo -n "Enter text: "
read text
echo -n "Enter pattern: "
read pattern
kmp_search "$text" "$pattern"
```

50. Floyd-Warshall Algorithm (All-Pairs Shortest Path)

```
#!/bin/bash
floyd_warshall() {
  local n=$1
  declare -A graph

  for ((i=0; i<n; i++)); do
    for ((j=0; j<n; j++)); do
```

```
      echo -n "Enter distance from $i to $j (999 if no path): "
      read graph[$i,$j]
    done
  done

  for ((k=0; k<n; k++)); do
    for ((i=0; i<n; i++)); do
      for ((j=0; j<n; j++)); do
        (( graph[$i,$j] > graph[$i,$k] + graph[$k,$j] )) && graph[$i,$j]=$(( graph[$i,$k] +
graph[$k,$j] ))
      done
    done
  done

  echo "Shortest distances between all pairs:"
  for ((i=0; i<n; i++)); do
    for ((j=0; j<n; j++)); do
      printf "%3d " "${graph[$i,$j]}"
    done
    echo
  done
}

echo -n "Enter number of nodes: "
read n
floyd_warshall $n
```