Questões de Recursividade

FÁCEIS

1-Assinale a alternativa que melhor representa seu objetivo.

```
const fun = (n) => {
      if (n==0) return '0'
      else if (n==1) return '1'
     else return 1+fun(Math.floor(n/2)) + (n%2).toString()
 }
```

a) Aproxima para o maio inteiro e transforma em string.
b) Representação decimal do binário.
c) Representação binária do inteiro.
d) Aproxima para o menor inteiro e transforma em string.

Resposta: c) Representação binária do inteiro.

2- Assinale a alternativa que representa o resultado obtido.

```
const fun = (fn, [x,…xs]) => {
      if (x===undefined) return [];
      else return [fn(x),…fun(fn,xs)];
};
console.log(fun([1,2,3,4,5]));
```

a) [1,4,9,16,25]
b) [2,3,4,5,6]
c) [2,4,6,8,10]
d) [1,2,3,4,5]

Resposta: c) [2,4,6,8,10]

3-O que é recursão em programação?
a) Um loop infinito.
b) Uma função que se chama repetidamente até atingir um caso base.
c) Uma função que só é usada em linguagens de baixo nível.
d) Um método que sempre retorna undefined.

Resposta: b) Uma função que se chama repetidamente até atingir um caso base.

4-Qual a importância de um caso base em recursão?
a) Evitar que a função continue indefinidamente.
b) Garantir que a função seja iterativa.
c) Garantir que a função use variáveis globais.
d) Melhorar a legibilidade do código.

Resposta: a) Evitar que a função continue indefinidamente.

5-O que acontece se uma função recursiva não tiver um caso base?
a) A função será chamada apenas uma vez.
b) Ela entrará em um loop infinito até consumir toda a memória.
c) A função nunca será chamada.
d) A função retornará automaticamente 0.

Resposta: b) Ela entrará em um loop infinito até consumir toda a memória.

6-Qual das opções abaixo é uma função recursiva para calcular o fatorial de um número?
a) function factorial(n) { return n * factorial(n-1); }
b) function factorial(n) { let result = 1; for(let i = 1; i <= n; i++) { result *= i; } return result; }
c) function factorial(n) { if (n <= 1) return 1; return n * factorial(n - 1); }
d) function factorial(n) { return n + factorial(n-1); }

Resposta: c) function factorial(n) { if (n <= 1) return 1; return n * factorial(n - 1); }

7-Escreva uma função recursiva para somar os números de um array. Qual é o caso base correto?
a) if (arr.length === 0) return 0;
b) if (arr[0] === null) return 0;
c) if (arr.length > 0) return 0;
d) if (arr.length === 1) return arr[0];

Resposta: a) if (arr.length === 0) return 0;

8-Qual é o caso base correto para calcular o fatorial recursivamente?
a) if (n === 0) return 0;
b) if (n <= 1) return 1;
c) if (n === 1) return 0;
d) if (n === -1) return n;

Resposta: b) if (n <= 1) return 1;

9-Em quais situações a recursão não é recomendada?
a) Quando o problema pode ser resolvido facilmente com laços iterativos.
b) Quando o caso base é desconhecido.
c) Quando a função precisa ser otimizada.
d) Quando o problema envolve processamento de texto.

Resposta: a) Quando o problema pode ser resolvido facilmente com laços iterativos.

10-Qual das seguintes é uma vantagem da recursão sobre laços iterativos?
a) Ela sempre executa mais rápido que laços.
b) Ela pode ser mais expressiva e simples para resolver problemas que envolvem subdivisão, como árvores.
c) Ela nunca consome mais memória do que a iteração.
d) Ela não precisa de um caso base.

Resposta: b) Ela pode ser mais expressiva e simples para resolver problemas que envolvem subdivisão, como árvores.

MÉDIAS

1-Assinale a alternativa que melhor representa seu objetivo.

```
const fun = (a,b) => {
    if (b==0) return 0
    else return fun(a,b-1)+a
}
```

a) Multiplica dois números naturais.
b) Somas dois números naturais.
c) Potencia a^b.
d) Potencia b^a.

Resposta: a) Multiplica dois números naturais.

2- Como calcular o fatorial de um número natural?
a) const fun = (n) => n==1 ? 1: fun(n-1)
b) const fun = (n) => n==1 ? 1: fun(n-1)*n
c) const fun = (n) => n==0 ? 1: fun(n-1)*n
d) const fun = (n) => n==0 ? 1: fun(n-1)

Resposta: c) const fun = (n) => n==0 ? 1: fun(n-1)*n

3- Função recursiva que verifica se uma lista de inteiros está ordenada em ordem crescente. Qual trecho?

```
const fun = ([x,y,...xs]) => {
    if (y === undefined) return true;
    else if (x>y) return false;
    else return fun(B);
};
```

a) xs
b) [y,...xs]
c) xs.slice(1)
d) [x,...xs]

Resposta: b) [y,...xs]

4- Assinale a alternativa que representa o resultado obtido.

```
const fun = (n) => {
      if (n>=101) return n-10
      else return fun(fun(n+11))
}
console.log(fun(98))
```

a) 88
b) 91
c) 99
d) 101

Resposta: b) 91

5- Assinale a alternativa que representa o resultado obtido.

```
const fun = ([x,y,…xs]) => {
      if (y===undefined) return [];
      else return [x+y,…fun(xs)];
};
console.log(fun([1,2,3,4,5]));
```

a) [3,7]
b) [3,5,7,9]
c) [3,7,5]
d) [15]

Resposta: a) [3,7]

6-Qual é a saída da seguinte função recursiva?

```
function sum(n) {
  if (n <= 0) return 0;
  return n + sum(n - 1);
}
console.log(sum(3));
```

a) 3
b) 6
c) 0
d) 1

Resposta: b) 6

7-Qual das seguintes funções calcula a soma de todos os elementos de uma lista de forma recursiva?
a) function sum(arr) { return arr.reduce((acc, x) => acc + x); }
b) function sum(arr) { return arr[0] + sum(arr.slice(1)); }
c) function sum(arr) { if (arr.length === 0) return 0; return arr[0] + sum(arr.slice(1)); }
d) function sum(arr) { let total = 0; for(let i = 0; i < arr.length; i++) { total += arr[i]; } return total; }

Resposta: c) function sum(arr) { if (arr.length === 0) return 0; return arr[0] + sum(arr.slice(1));}

8- Qual é o comportamento da função a seguir, e o que ela retorna para fib(6)?
```
function fib(n) {
  if (n <= 1) return n;
  return fib(n - 1) + fib(n - 2);
}
```
a) Ela calcula o fatorial de n e retorna 720.
b) Ela calcula o 6º número de Fibonacci e retorna 8.
c) Ela entra em um loop infinito, pois não tem caso base.
d) Ela calcula o 6º número de Fibonacci e retorna 13.

Resposta: b) Ela calcula o 6º número de Fibonacci e retorna 8.

9- Qual das opções descreve melhor o funcionamento da função recursiva abaixo?
```
function reverseString(str) {
  if (str === "") return "";
  return reverseString(str.slice(1)) + str[0];
}
```
a) Inverte uma string recursivamente, removendo o primeiro caractere em cada chamada e adicionando-o no final.
b) Adiciona o primeiro caractere da string ao final de uma string vazia em cada chamada recursiva.
c) Remove o último caractere da string em cada chamada até que não haja mais caracteres.
d) Retorna a string original sem alterações.

Resposta: a) Inverte uma string recursivamente, removendo o primeiro caractere em cada chamada e adicionando-o no final.

10-Dada a seguinte função recursiva, qual é a função da linha return 1;?
```
function factorial(n) {
  if (n === 0) return 1;
  return n * factorial(n - 1);
}
```
a) Retornar 1 como o valor final de todos os cálculos recursivos.
b) Garantir que a função tenha um caso base e não entre em loop infinito.

c) Garantir que a função retorne o valor correto ao calcular o fatorial de 1.
d) Substituir o valor de n por 1 em todas as chamadas.

Resposta: b) Garantir que a função tenha um caso base e não entre em loop infinito.

DIFÍCEIS

1-Assinale a alternativa que melhor representa seu objetivo.

```
const fun = (n) => {
     if (n==1) return 0
     else return 1+fun(Math.floor(n/2))
}
```

a) Retorna a quantidade de zeros de um número inteiro.
b) Calcula o log2n.
c) Retorna a quantidade de algarismos de um número inteiro.
d) Calcula a raiz de n.

Resposta: b) Calcula o log2n.

2- Assinale a alternativa que melhor representa seu objetivo.

```
const fun = (n) => {
   const helper = (a) => (b) =>{
      if (b==1) return true
      else if ((a%b)==0) return false
      else return helper(a) (b-1)
}
     if (n==1) return false
     else return helper(n) (Math.floor(n/2))
}
```

a) Retorna o maior divisor de um número inteiro.
b) Retorna o maior fator primo .
c) Testa se um inteiro é divisível por 2.
d) Testa se um inteiro é primo.

Resposta: d) Testa se um inteiro é primo.

3-Assinale a alternativa que melhor representa seu objetivo.

```
const fun = ([x, ...xs], [y, ...ys]) => {
     if (x === undefined || y === undefined)
return [];
     else return [[x,y],...fun(xs,ys)];
}
```

a) Retorna a soma dos pares das duas listas.

b) Retorna a concatenação dos elementos das duas listas.
c) Retorna uma lista de pares, combinando.
d) Retorna os elementos comuns entre as duas listas.

Resposta: c) Retorna uma lista de pares, combinando.

4- Assinale a alternativa que melhor representa seu objetivo.

```
const fun = ([x, ...xs],z) => {
    if (z === 0) return true;
    else if (x === undefined || z < 0) return false;
    else return fun(xs,z-x) || fun(xs,z);
}
```

a) Retorna todos os subconjuntos que somam o valor z.
b) Verifica se existe um subconjunto cujos elementos somam o valor z.
c) Retorna a soma de todos os subconjuntos que podem ser formados.
d) Verifica se todos os elementos da lista são menores que z.

Resposta: b) Verifica se existe um subconjunto cujos elementos somam o valor z.

5- Calcular o mdc entre 2 números inteiros. Qual o trecho A e o trecho B, respectivamente?

```
const fun = (a,b) => {
    if (a==b) return a
    else if (a>b) return A
    else return B
}
```

a) fun(a,a-b) e fun(b,b-a)
b) fun(a,b-a) e fun(b-a,a)
c) fun(a,b) e fun(b,a)
d) fun(a-b-b) e fun(a,b-a)

Resposta: d) fun(a-b-b) e fun(a,b-a)

6- Como remover todos os elementos pares de uma lista de inteiros?

```
const fun = ([x,...xs]) => {
    if (y === undefined) return [];
    else if A return fun(xs);
    else return [x, ...fun(xs)];
};
```

a) (x%2===0)
b) (x/2===0)
c) (x%2===1)
d) (x/2===1)

Resposta: a) (x%2===0)

7- Assinale a alternativa que representa o resultado obtido.

```
const fun = (n) => {
      if (n==0) return 0
      else return 1+fun(Math.floor(n/10))
}
console.log(fun(4887655303))
```

a) 4
b) 10
c) 49
d) 4000000000

Resposta: b) 10

8-Qual é a complexidade de tempo esperada para a função recursiva a seguir que calcula a soma de todos os números de uma lista?

```
function sum(arr) {
  if (arr.length === 0) return 0;
  return arr[0] + sum(arr.slice(1));
}
```

a) O(n), pois a função percorre todos os elementos do array uma vez.
b) O(n²), pois cada chamada de slice() cria uma nova cópia do array.
c) O(log n), pois a recursão reduz pela metade o número de elementos a cada chamada.
d) O(1), pois a função termina em uma única chamada recursiva.

Resposta: b) O(n²), pois cada chamada de slice() cria uma nova cópia do array.

9-A seguinte função recursiva é uma tentativa de verificar se uma string é um palíndromo. Qual é o erro lógico nesse código?

```
function isPalindrome(str) {
  if (str.length <= 1) return true;
  if (str[0] !== str[str.length - 1]) return false;
  return isPalindrome(str.slice(1, -1));
}
```

a) O caso base está incorreto, pois não inclui strings vazias.
b) A função não lida corretamente com strings de comprimento ímpar.
c) O uso de slice(1, -1) pode criar substrings incorretas ao trabalhar com strings longas.
d) Não há erro lógico no código; ele funciona corretamente.

Resposta: d) Não há erro lógico no código; ele funciona corretamente.

10-O código abaixo implementa uma função recursiva para ordenar um array usando o algoritmo de quick sort. Qual é o maior problema de desempenho potencial?

```
function quickSort(arr) {
  if (arr.length <= 1) return arr;
  let pivot = arr[0];
  let left = arr.slice(1).filter(x => x < pivot);
  let right = arr.slice(1).filter(x => x >= pivot);
  return [...quickSort(left), pivot, ...quickSort(right)];
}
```

a) A função não possui um caso base adequado.
b) O uso de filter() e slice() aumenta a complexidade de tempo para $O(n^2)$.
c) O algoritmo quick sort não pode ser implementado recursivamente.
d) A função não funciona corretamente se houver valores duplicados no array.

Resposta: b) O uso de filter() e slice() aumenta a complexidade de tempo para $O(n^2)$.