Actividad 3

Erick Gonzalez Parada ID: 178145
Matemáticas discretas, Universidad de las Américas Puebla, Puebla, México

September 20, 2023

Abstract

Demostración por inducción e invariante de ciclo

Keywords: inducción, ciclo

1 Demostración por inducción

$$n! \geq 2^n - 1 \tag{1}$$

Se probara que la ecuación 1 es verdadera por inducción

Paso base

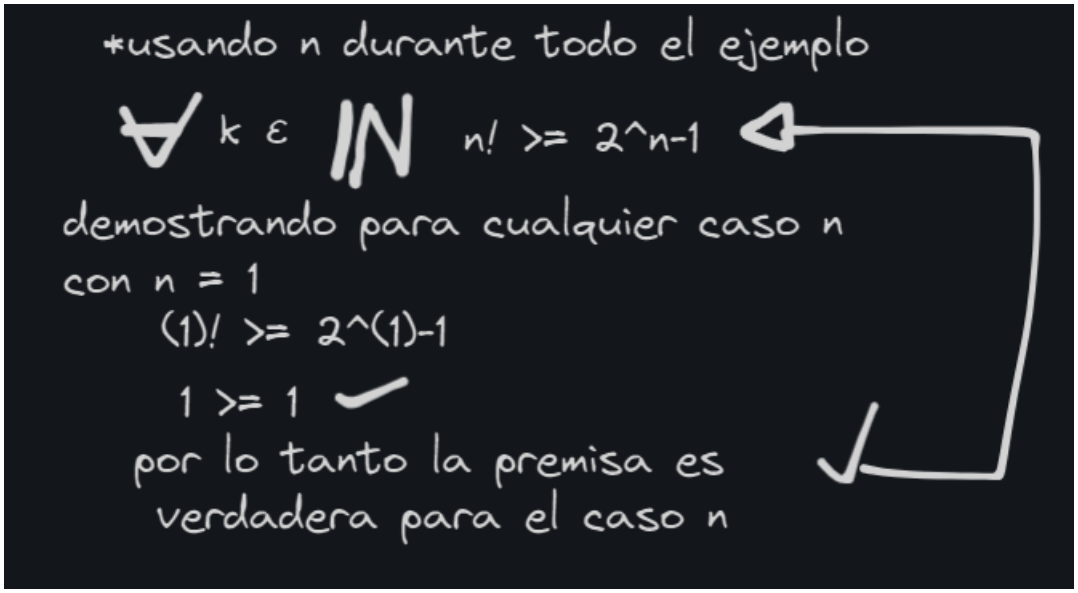

Figura 1: paso base

Paso inductivo

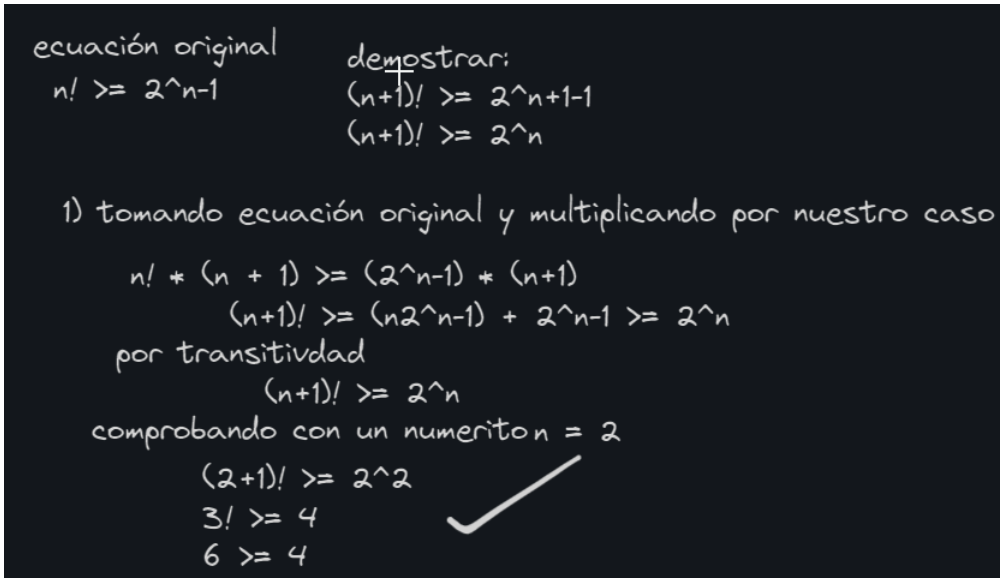

Figura 2: paso inductivo

conclusión parte 1

La forma inductiva matemática de demostrar que una ecuación es verdadera consiste en básicamente hacer compatible por medio del álgebra el paso base (fig 1) con el paso inductivo (fig 2).

2 Invariante de ciclo merge sort

Este algoritmo en pocas palabras divide recursivamente el problema hasta unidades individuales, luego las une de forma ordenada hacia atrás de forma recursiva.

código

```
#include <iostream>
#include <vector>

void merge(std::vector<int>& arr, int left, int middle, int right) {
    int n1 = middle - left + 1;
    int n2 = right - middle;

    // Crear subarreglos temporales
    std::vector<int> L(n1);
    std::vector<int> R(n2);

    // Copiar datos a los subarreglos L[] y R[]
    for (int i = 0; i < n1; i++) {
        L[i] = arr[left + i];
    }
    for (int j = 0; j < n2; j++) {
        R[j] = arr[middle + 1 + j];
    }
```

```
    // Fusionar los subarreglos en arr[left..right]
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    // Copiar los elementos restantes de L[], si los hay
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    // Copiar los elementos restantes de R[], si los hay
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(std::vector<int>& arr, int left, int right) {
    if (left < right) {
        int middle = left + (right - left) / 2;

        // Ordenar la primera mitad y la segunda mitad
        mergeSort(arr, left, middle);
        mergeSort(arr, middle + 1, right);

        // Combinar las mitades ordenadas
        merge(arr, left, middle, right);
    }
}

int main() {
    std::vector<int> arr = {12, 11, 13, 5, 6, 7};
    int arrSize = arr.size();

    std::cout << "Arreglo original: ";
    for (int i = 0; i < arrSize; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    mergeSort(arr, 0, arrSize - 1);
```

```
    std::cout << "Arreglo ordenado: ";
    for (int i = 0; i < arrSize; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Invariante de ciclo

esta la encontramos (en el caso del código) en antepenúltimo while loop, básicamente el más importante del algoritmo, en donde juntamos el array izquierdo ya ordenado con el array derecho, por ende la invariante de ciclo es esa: mientras estamos juntando en un array "final" o "resultante" array[left] & array[right] están ordenando, teniendo así, la solución parcial ya que nuestro objetivo es ordenar todo.

3 Invariante de ciclo en algoritmo erróneo

algoritmo incorrecto

En el siguiente algoritmo (código en python) se forzó un error y se va analizar que le sucede que a la invariante de ciclo.

```
  def accSum(list):
    for element in list:
        sum = 0
        sum = sum + element
    return sum

nums = [1,2,3]
print(accSum(nums))
```

Para los principiantes programando quizá no sea tan obvio el error al principio pero es muy sencillo, nuestra invariante de ciclo es que lo que ya hemos explorado en el array ya tenemos la suma parcial de eso, por ende tenemos la solución parcial aquí significan básicamente la misma cosa.
Sin embargo debido a un error de semántica en la implementación del algoritmo alteramos gravemente nuestra invariante ya que estamos reseteando nuestra variable donde guardamos esta promesa de la invariante que es que siempre tendremos la suma (y solución) parcial presente.

algoritmo correcto

```
  def accSum(list):
  sum = 0
    for element in list:
        sum = sum + element
    return sum

nums = [1,2,3]
print(accSum(nums))
```