Instituto Politécnico Nacional

Escuela Superior de Computo

Ejercicio 03

***"Simulación Producto2Mayores"***

Mora Ayala José Antonio

Análisis de Algoritmos



Para el algoritmo analizado por casos en clase y video lección "Producto2Mayores", realice la simulación de su mejor, peor y caso medio; realizando las modificaciones y adaptaciones necesarias para verificar los tres casos en n=2500 y n=5000 considerando al menos 10,000 iteraciones del algoritmo con cada n y diferente distribución de los números.

Para el mejor caso basta con tener un archivo de números que coloque en los dos primeros números a los dos mayores

Para el peor caso basta con tener un archivo ordenado ascendentemente para cada n

Para el caso medio se deberán de hacer al menos 10,000 iteraciones para cada n generando arreglos aleatorios en cada iteración y comprobar el número de operaciones básicas promedios totales para enfrentarlas al modelo del caso medio.

* Incluir portada con los de datos del alumno, datos del trabajo y fotografía del alumno
* Incluir tabla comparativa de los resultados teóricos y prácticos para n=2,500 y n=5,000
* Enmarcar los códigos o pseudocódigos de cada algoritmo en el documento y manejar formato de colores
* Recordar manejar encabezados y pies de página.
* Entregar reporte de simulación y códigos y archivos de números empleados en un archivo comprimido.

## Caso NUMERO 1 (mejor caso)

|  |
| --- |
| 1. int Producto2Mayores(int \*A, int n) 2. { 3. int aux = 0; 4. int mayor1, mayor2, i; 5. if (A[1] > A[2]) 6. { 7. mayor1 = A[1]; 8. mayor2 = A[2]; 9. // Mejor Caso 10. aux += 3; //3 operaciones Comparacion + 2 asignaciones 11. } 12. else 13. { 14. mayor1 = A[2]; 15. mayor2 = A[1]; 16. aux += 3; //operaciones Comparacion + 2 asignaciones 17. } 18. i = 3; 19. while (i <= n) 20. { 21. if (A[i] > mayor1) 22. { 23. mayor2 = mayor1; 24. mayor1 = A[i]; 25. aux += 3; //operaciones Comparacion + 2 asignaciones 26. } 27. else if (A[i] > mayor2) 28. { 29. mayor2 = A[i]; 30. aux += 3; //operaciones Comparacion,   comparacion en falso + 1 asignacion 31. } 32. else 33. { 34. aux += 2; //2 comparaciones en falso 35. } 36. i = i + 1; 37. } 38. printf("%d", aux); 39. return (mayor1 \* mayor2); 40. } |

## Archivo de Entrada

Números.txt

|  |
| --- |
| 1. 2019439 2. 1846873530 3. 1799388853 4. 207442309 5. 494387258 6. 654552922 7. 834068396 8. 856834115 9. 870128690 10. 966245083 11. 1511588122 |

**Función de Complejidad Temporal   
­­**

Para realizar el análisis del mejor caso se copian los valores que se muestran en el archivo Numeros.txt al arreglo de tamaño n (n debe ser menor a 11, debido a que no tenemos más elementos dentro de ese archivo), posterior a eso el algoritmo comienza

## Peor caso

|  |
| --- |
| 1. int Producto2Mayores(int \*A, int n) 2. { 3. int aux = 0; 4. int mayor1, mayor2, i; 5. if (A[1] > A[2]) 6. { 7. mayor1 = A[1]; 8. mayor2 = A[2]; 9. // Mejor Caso 10. aux += 3; //3 operaciones Comparacion + 2 asignaciones 11. } 12. else 13. { 14. mayor1 = A[2]; 15. mayor2 = A[1]; 16. aux += 3; //operaciones Comparacion + 2 asignaciones 17. } 18. i = 3; 19. while (i <= n) 20. { 21. if (A[i] > mayor1) 22. { 23. mayor2 = mayor1; 24. mayor1 = A[i]; 25. aux += 3; //operaciones Comparacion + 2 asignaciones 26. } 27. else if (A[i] > mayor2) 28. { 29. mayor2 = A[i]; 30. aux += 3; //operaciones Comparacion,   comparacion en falso + 1 asignacion 31. } 32. else 33. { 34. aux += 2; //2 comparaciones en falso 35. } 36. i = i + 1; 37. } 38. printf("%d", aux); 39. return (mayor1 \* mayor2); 40. } |

## Archivo de Texto

|  |
| --- |
| 1. 10 2. 20 3. 30 4. 40 5. 50 6. 60 7. 70 8. 80 9. 90 10. 100 11. 110 12. 120 13. 130 14. 140 15. 150 16. 160 17. 170 18. 180 19. 190 20. 200 |

**Función de Complejidad Temporal**

## bucle numero 3

|  |
| --- |
| 1. //Función que recibe un arreglo y un tamaño de datos 2. int Producto2Mayores(int \*A, int n) 3. { 4. int mayor1, mayor2, i,k,j, promedio = 0, aux; 5. int repeticiones = 10000; 6. srand(time(NULL)); 7. //Ciclo que repetira el ciclo con respecto a la cantidad 8. //de repeticiones indicadas, para aumentar veracidad del 9. //resultado 10. for (k = 0; k < repeticiones; k++) 11. { 12. aux = 0; 13. //Realizando la asignación de números aleatorios  dentro del arreglo 14. for (j = 0; j < n; j++) 15. { 16. A[j] = 10000 + rand() % (30001 + 10000); 17. } 18. if (A[1] > A[2]) 19. { 20. mayor1 = A[1]; 21. mayor2 = A[2]; 22. aux += 3; //operaciones Comparacion   + 2 asignaciones 23. }else 24. { 25. mayor1 = A[2]; 26. mayor2 = A[1]; 27. aux += 3; //operaciones Comparacion   + 2 asignaciones 28. } 29. i = 3; 30. while (i <= n) 31. { 32. if (A[i] > mayor1) 33. { 34. mayor2 = mayor1; 35. mayor1 = A[i]; 36. aux += 3; //operaciones Comparacion   + 2 asignaciones 37. } 38. else if (A[i] > mayor2) 39. { 40. mayor2 = A[i]; 41. aux += 2; //2 operaciones   Comparacion + 1 asignaciones 42. } 43. else 44. { 45. aux += 2; //2 operaciones Comparacion en falso 46. } 47. i = i + 1; 48. } 49. promedio += aux; //Llevando la cuenta del   contador al termino de cada ideracion en el ciclo 50. } 51. promedio = promedio / repeticiones; 52. printf("%d", promedio); 53. return (mayor1 \* mayor2); 54. } |

**Función de Complejidad Temporal:**

|  |  |  |
| --- | --- | --- |
| n | Teórico | Práctico |
| 2500 | 6664 | 5005 |
| 5000 | 13,331 | 10,006 |