Assignment - 07

Team members:
Mahitha Gurrala(U19CS066)
Guntuboina Venkata Sankirtana (U19CS068)
Krithikha Balamurugan (U19CS076)

1.1 Find a computational problem that you can solve using the greedy approach.

WIGGLE SUBSEQUENCE

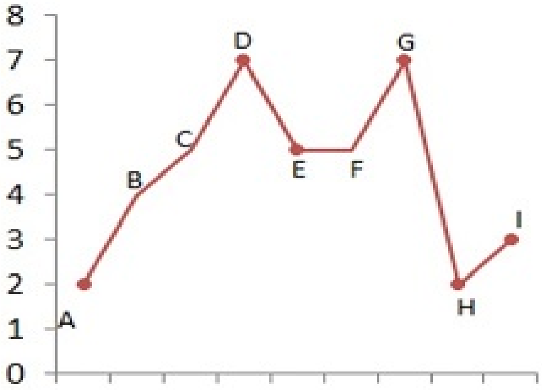
Problem statement: **Given an integer array Arr, return *the length of the longest wiggle subsequence of* Arr.**

Explanation:

A wiggle sequence is a sequence where the differences between successive numbers strictly alternate between positive and negative. The first difference (if one exists) may be either positive or negative. A sequence with two or fewer elements is trivially a wiggle sequence.

- For example, [1, 7, 4, 9, 2, 5] is a wiggle sequence because the differences (6, -3, 5, -7, 3) alternate between positive and negative.
- In contrast, [1, 4, 7, 2, 5] and [1, 7, 4, 5, 5] are not wiggle sequences. The first is not because its first two differences are positive, and the second is not because its last difference is zero.

A subsequence is obtained by deleting some elements (possibly zero) from the original sequence, leaving the remaining elements in their original order.



Sample Test Cases:

Example 1:

Input: nums = [1,7,4,9,2,5]
Output: 6
Explanation: The entire sequence is a wiggle sequence with differences (6, -3, 5, -7, 3).

Example 2:

Input: nums = [1,17,5,10,13,15,10,5,16,8]
Output: 7
Explanation: There are several subsequences that achieve this length.
One is [1, 17, 10, 13, 10, 16, 8] with differences (16, -7, 3, -3, 6, -8).

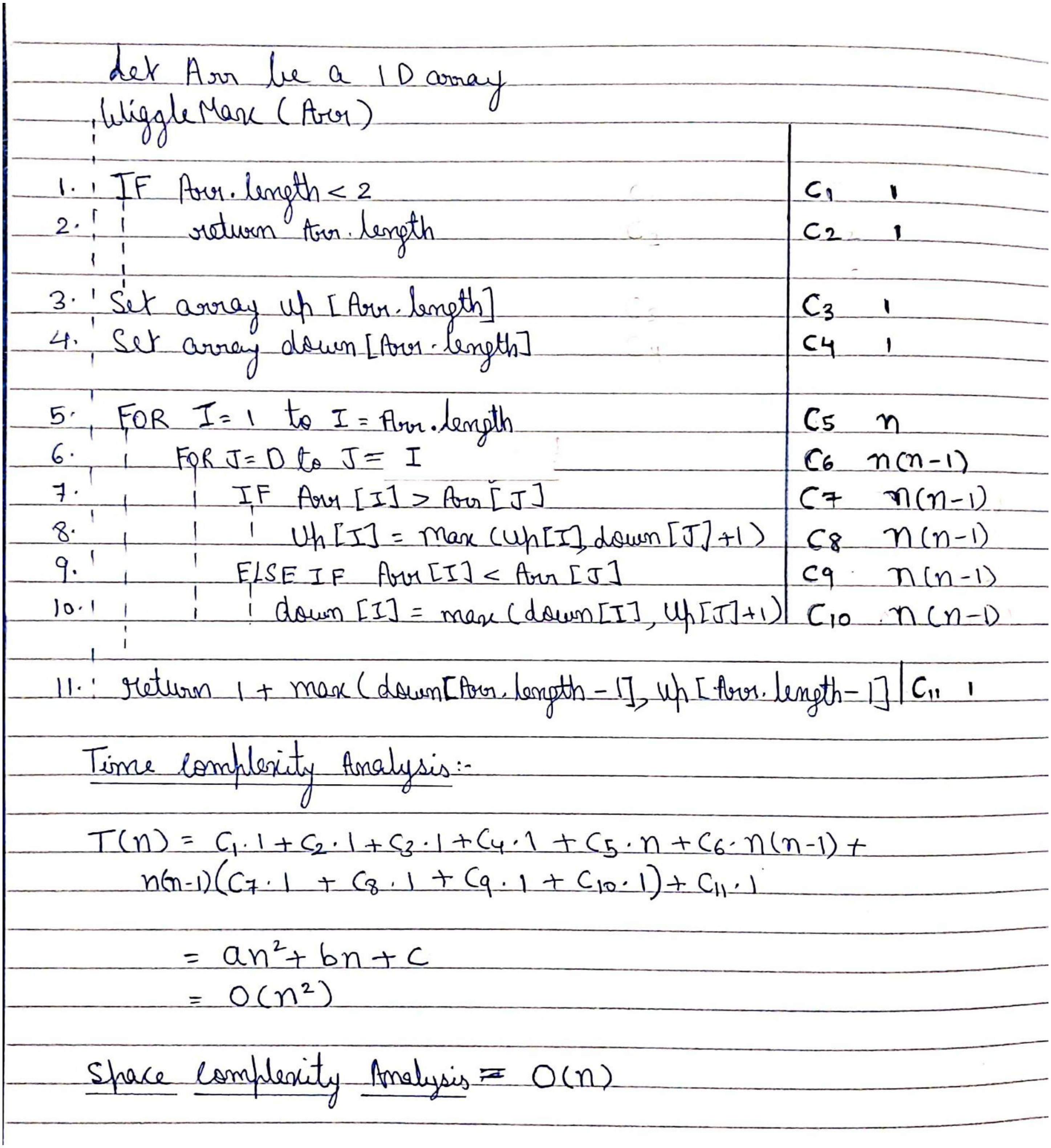
Example 3:

Input: nums = [1,2,3,4,5,6,7,8,9]
Output: 2

1.2. (T) Write pseudocodes to design the algorithms for the above mentioned computational problem using the Greedy approach as well as dynamic programming.

1.3. (T) Analyze the time complexity of above algorithms.

Dynamic Programming:



```
Let Arr be a 1D array
WiggleMax (Arr)

1.  IF Arr.length < 2                                          C1    1
2.      return Arr.length                                      C2    1

3.  Set array up [Arr.length]                                  C3    1
4.  Set array down [Arr.length]                                C4    1

5.  FOR I = 1 to I = Arr.length                                C5    n
6.      FOR J = 0 to J = I                                     C6    n(n-1)
7.          IF Arr[I] > Arr[J]                                 C7    n(n-1)
8.              up[I] = Max (up[I], down[J]+1)                 C8    n(n-1)
9.          ELSE IF Arr[I] < Arr[J]                            C9    n(n-1)
10.             down[I] = max (down[I], up[J]+1)               C10   n(n-1)

11. return 1 + max (down[Arr.length - 1], up[Arr.length - 1]]  C11   1
```

Time complexity Analysis :-

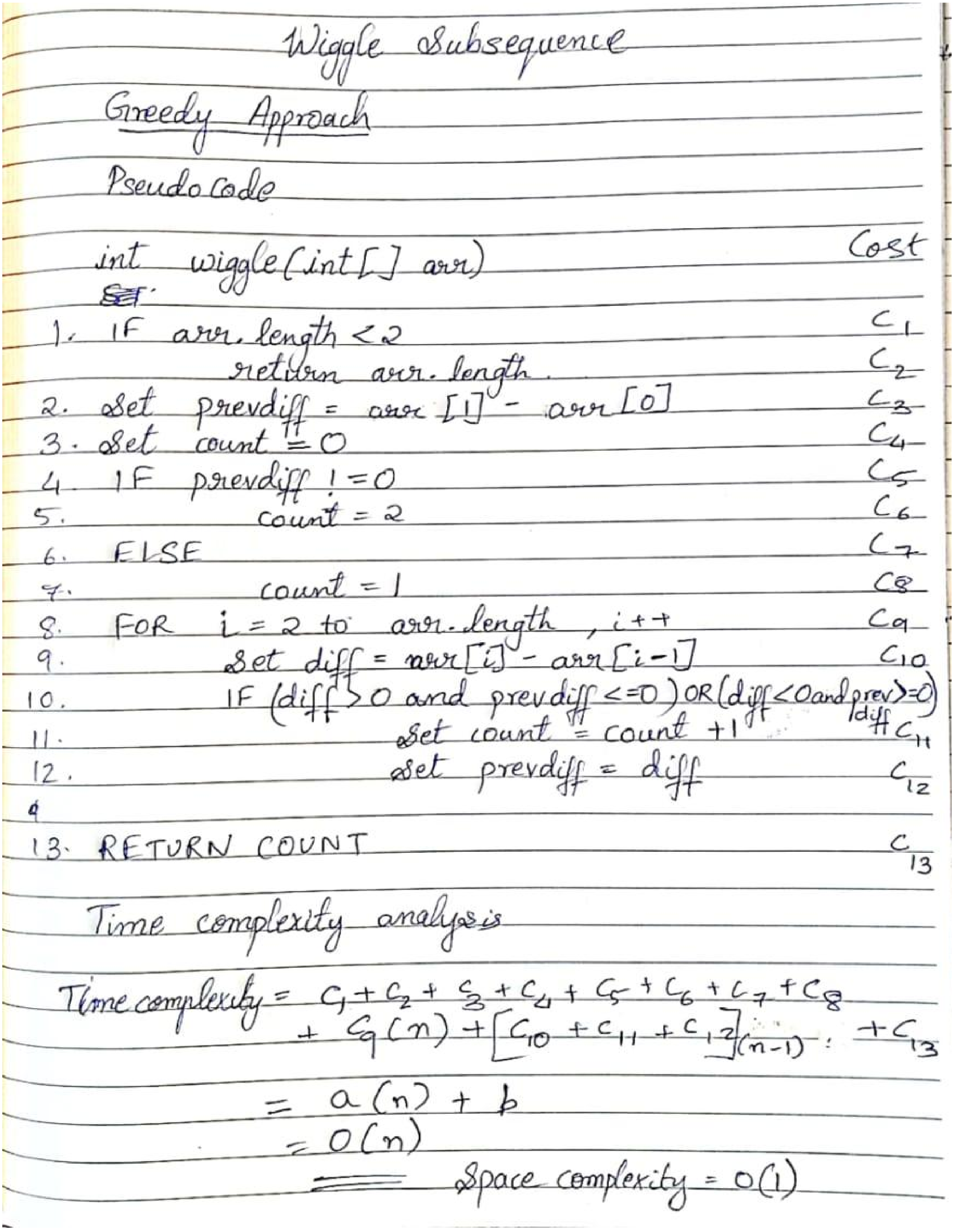
$$T(n) = C_1 \cdot 1 + C_2 \cdot 1 + C_3 \cdot 1 + C_4 \cdot 1 + C_5 \cdot n + C_6 \cdot n(n-1) + n(n-1)(C_7 \cdot 1 + C_8 \cdot 1 + C_9 \cdot 1 + C_{10} \cdot 1) + C_{11} \cdot 1$$

$$= an^2 + bn + c$$

$$= O(n^2)$$

Space complexity Analysis = $O(n)$

Greedy Approach:



Wiggle Subsequence

Greedy Approach

Pseudo Code

```
int wiggle(int[] arr)                                          Cost
~~SET~~
IF arr.length < 2                                              $c_1$
        return arr.length.                                     $c_2$
Set prevdiff = arr[1] - arr[0]                                 $c_3$
Set count = 0                                                  $c_4$
IF prevdiff != 0                                               $c_5$
        count = 2                                              $c_6$
ELSE                                                           $c_7$
        count = 1                                              $c_8$
FOR i = 2 to arr.length , i++                                  $c_9$
        Set diff = arr[i] - arr[i-1]                           $c_{10}$
        IF (diff > 0 and prevdiff <= 0) OR (diff < 0 and prevdiff >= 0)
                Set count = count + 1                          $c_{11}$
                Set prevdiff = diff                            $c_{12}$
RETURN COUNT                                                   $c_{13}$
```

Time complexity analysis

Time complexity $= c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7 + c_8 + c_9(n) + [c_{10} + c_{11} + c_{12}]_{(n-1)} + c_{13}$

$= a(n) + b$

$= O(n)$

Space complexity $= O(1)$

1.4 Provide the details of Hardware/Software you used to implement algorithms and to measure the time.

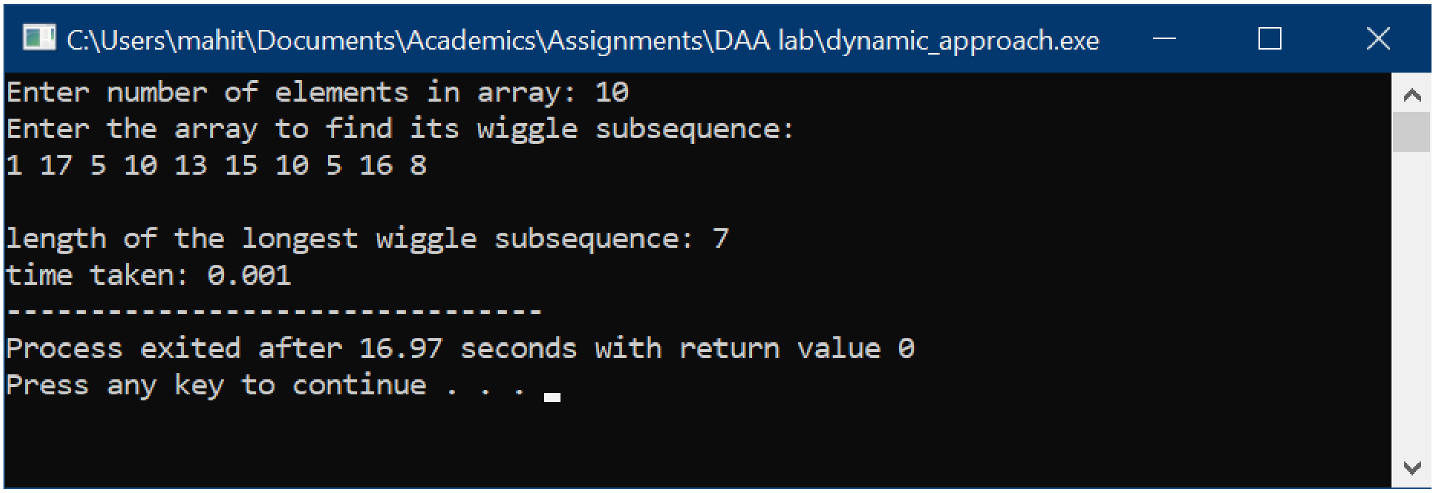
Compiler	Dev C++ 5.11
OS Name	Microsoft Windows 10 Home (i5 8th Gen)
Version	10.0.19042 Build 19042
System Name	DESKTOP-BLE6CMQ
System Model	HP Pavilion x360 Convertible 14-ba1xx
System Type	x64-based PC
Processor	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 1800 Mhz, 4 Core(s), 8 Logical Processor(s)
BIOS Version/Date	Insyde F.54, 04-12-2019
Installed Physical Memory (RAM)	8.00 GB
Total Physical Memory	7.88 GB
Available Physical Memory	1.75 GB
Total Virtual Memory	12.4 GB
Available Virtual Memory	4.59 GB
Page File Space	4.50 GB

1.5 Implement the above algorithms and submit the code (complete programs).

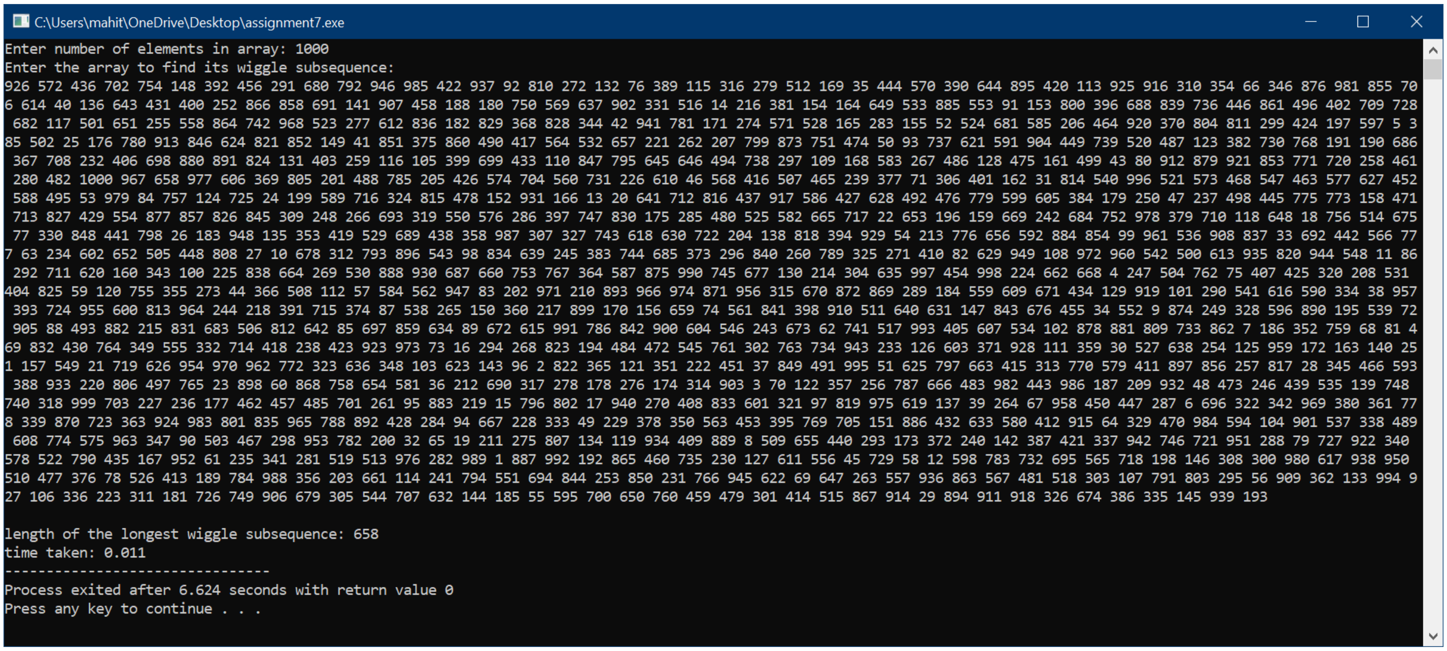
Dynamic Programming

```
#include <iostream>
#include <time.h>
using namespace std;
clock_t begin, end;
double time_;
int wiggleMax(int arr[], int n){
    if (n<2){
        return n;
    }
    int up[n]={0}, down[n]={0};
    int i,j;
    for(i=1; i<n; i++){
        for(j=0; j<i;j++){
            if(arr[i]>arr[j]){
                up[i]=max(up[i], down[j]+1);
            }
            else if(arr[i]<arr[j]){
                down[i]=max(down[i], up[j]+1);
            }
        }
    }
    return 1+max(down[n-1], up[n-1]);
}

int main(){
    int n;
    cout<<"Enter number of elements in array: ";
    cin>>n;
    cout<<"Enter the array to find its wiggle subsequence: ";
    int i;
    int arr[n];
    for(i=0; i<n; i++){
        cin>>arr[i];
    }

    cout<<endl<<"length of the longest wiggle subsequence: ";
    begin = clock();
    cout <<  endl<<"\nMaximum length of subsequence: " << wiggleMax(arr, n) << endl;
    end = clock();
    time_ = ((double)(end-begin)) / CLOCKS_PER_SEC;
    cout<<"\nTime taken :"<<time_;
    return 0;}
```

Output:

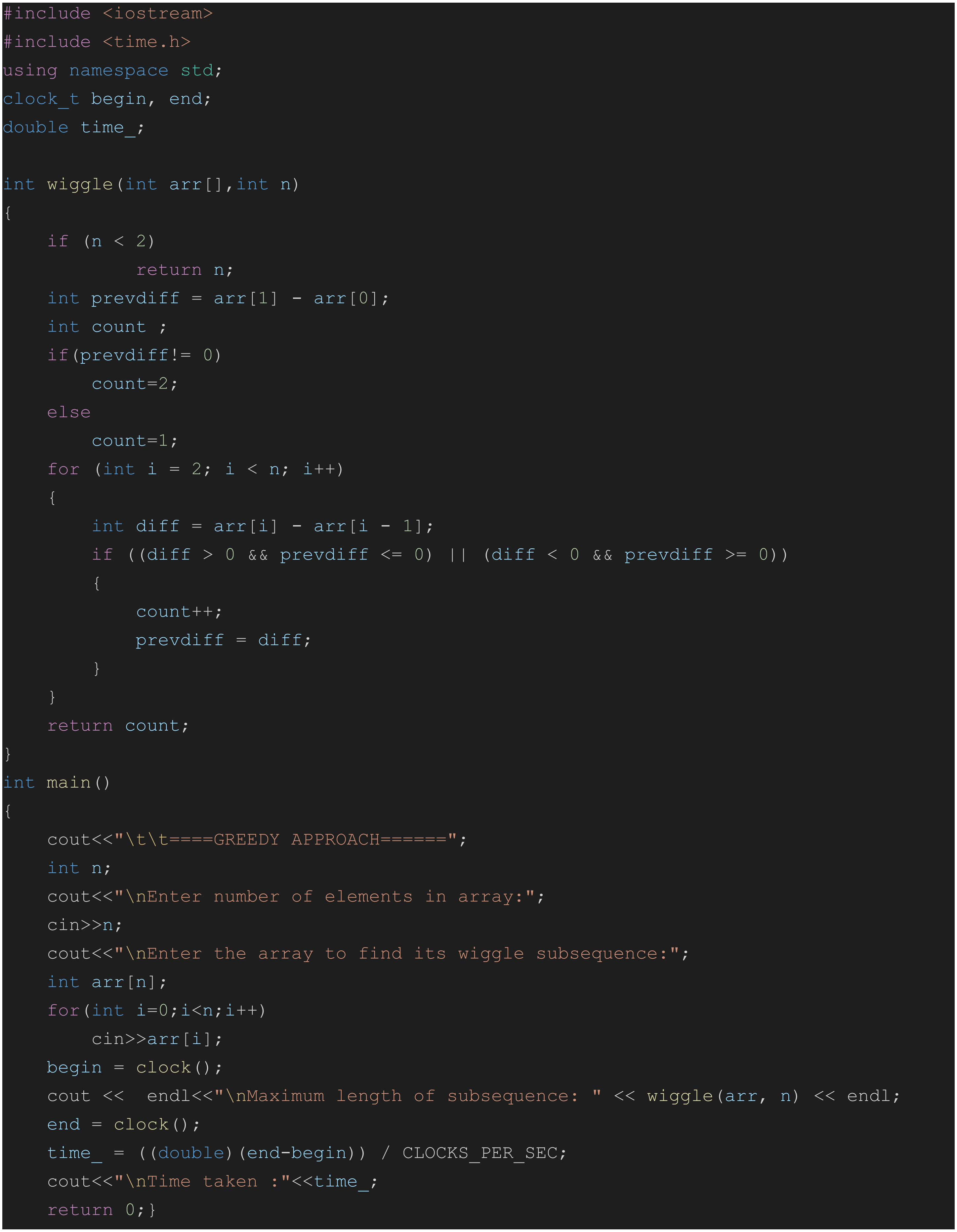

C:\Users\mahit\Documents\Academics\Assignments\DAA lab\dynamic_approach.exe

```
Enter number of elements in array: 10
Enter the array to find its wiggle subsequence:
1 17 5 10 13 15 10 5 16 8

length of the longest wiggle subsequence: 7
time taken: 0.001
--------------------------------
Process exited after 16.97 seconds with return value 0
Press any key to continue . . .
```



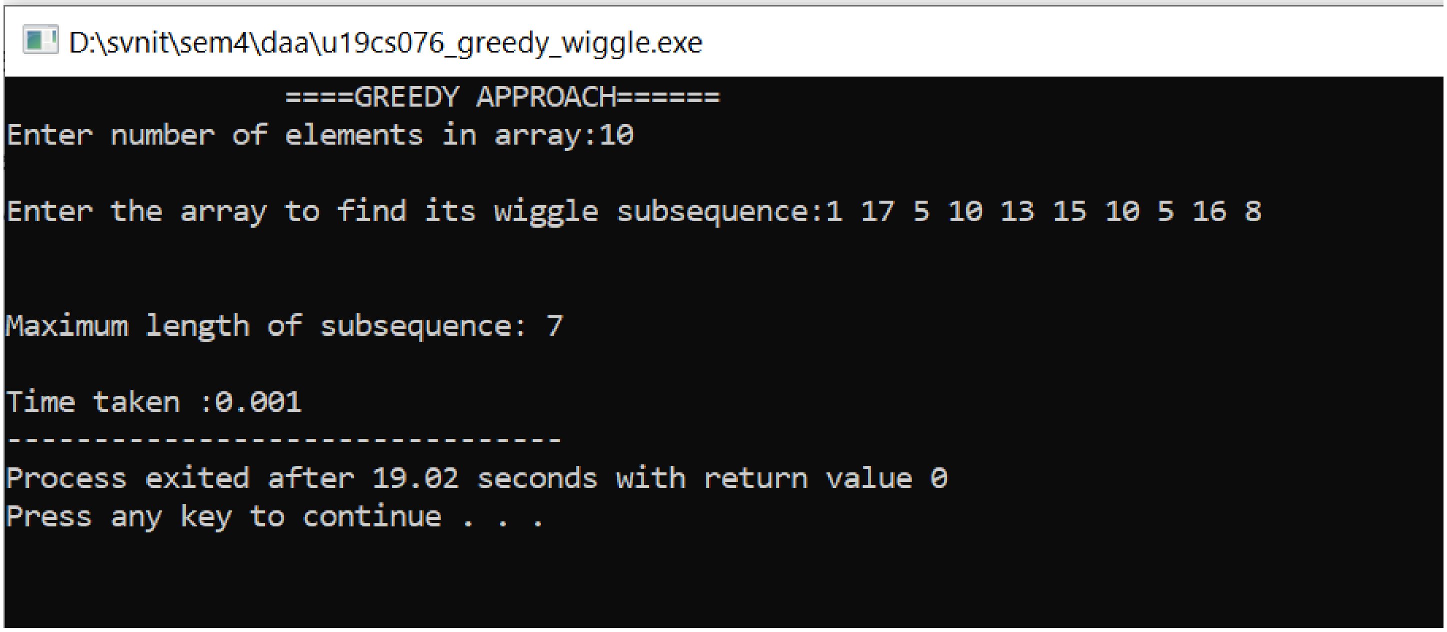
```
Enter number of elements in array: 1000
Enter the array to find its wiggle subsequence:
926 572 436 702 754 148 392 456 291 680 792 946 985 422 937 92 810 272 132 76 389 115 316 279 512 169 35 444 570 390 644 895 420 113 925 916 310 354 66 346 876 981 855 70
6 614 40 136 643 431 400 252 866 858 691 141 907 458 188 180 750 569 637 902 331 516 14 216 381 154 164 649 533 885 553 91 153 800 396 688 839 736 446 861 496 402 709 728
 682 117 501 651 255 558 864 742 968 523 277 612 836 182 829 368 828 344 42 941 781 171 274 571 528 165 283 155 52 524 681 585 206 464 920 370 804 811 299 424 197 597 5 3
85 502 25 176 780 913 846 624 821 852 149 41 851 375 860 490 417 564 532 657 221 262 207 799 873 751 474 50 93 737 621 591 904 449 739 520 487 123 382 730 768 191 190 686
 367 708 232 406 698 880 891 824 131 403 259 116 105 399 699 433 110 847 795 645 646 494 738 297 109 168 583 267 486 128 475 161 499 43 80 912 879 921 853 771 720 258 461
 280 482 1000 967 658 977 606 369 805 201 488 785 205 426 574 704 560 731 226 610 46 568 416 507 465 239 377 71 306 401 162 31 814 540 996 521 573 468 547 463 577 627 452
 588 495 53 979 84 757 124 725 24 199 589 716 324 815 478 152 931 166 13 20 641 712 816 437 917 586 427 628 492 476 779 599 605 384 179 250 47 237 498 445 775 773 158 471
 713 827 429 554 877 857 826 845 309 248 266 693 319 550 576 286 397 747 830 175 285 480 525 582 665 717 22 653 196 159 669 242 684 752 978 379 710 118 648 18 756 514 675
 77 330 848 441 798 26 183 948 135 353 419 529 689 438 358 987 307 327 743 618 630 722 204 138 818 394 929 54 213 776 656 592 884 854 99 961 536 908 837 33 692 442 566 77
7 63 234 602 652 505 448 808 27 10 678 312 793 896 543 98 834 639 245 383 744 685 373 296 840 260 789 325 271 410 82 629 949 108 972 960 542 500 613 935 820 944 548 11 86
 292 711 620 160 343 100 225 838 664 269 530 888 930 687 660 753 767 364 587 875 990 745 677 130 214 304 635 997 454 998 224 662 668 4 247 504 762 75 407 425 320 208 531
404 825 59 120 755 355 273 44 366 508 112 57 584 562 947 83 202 971 210 893 966 974 871 956 315 670 872 869 289 184 559 609 671 434 129 919 101 290 541 616 590 334 38 957
 393 724 955 600 813 964 244 218 391 715 374 87 538 265 150 360 217 899 170 156 659 74 561 841 398 910 511 640 631 147 843 676 455 34 552 9 874 249 328 596 890 195 539 72
 905 88 493 882 215 831 683 506 812 642 85 697 859 634 89 672 615 991 786 842 900 604 546 243 673 62 741 517 993 405 607 534 102 878 881 809 733 862 7 186 352 759 68 81 4
69 832 430 764 349 555 332 714 418 238 423 923 973 73 16 294 268 823 194 484 472 545 761 302 763 734 943 233 126 603 371 928 111 359 30 527 638 254 125 959 172 163 140 25
1 157 549 21 719 626 954 970 962 772 323 636 348 103 623 143 96 2 822 365 121 351 222 451 37 849 491 995 51 625 797 663 415 313 770 579 411 897 856 257 817 28 345 466 593
 388 933 220 806 497 765 23 898 60 868 758 654 581 36 212 690 317 278 178 276 174 314 903 3 70 122 357 256 787 666 483 982 443 986 187 209 932 48 473 246 439 535 139 748
740 318 999 703 227 236 177 462 457 485 701 261 95 883 219 15 796 802 17 940 270 408 833 601 321 97 819 975 619 137 39 264 67 958 450 447 287 6 696 322 342 969 380 361 77
8 339 870 723 363 924 983 801 835 965 788 892 428 284 94 667 228 333 49 229 378 350 563 453 395 769 705 151 886 432 633 580 412 915 64 329 470 984 594 104 901 537 338 489
 608 774 575 963 347 90 503 467 298 953 782 200 32 65 19 211 275 807 134 119 934 409 889 8 509 655 440 293 173 372 240 142 387 421 337 942 746 721 951 288 79 727 922 340
578 522 790 435 167 952 61 235 341 281 519 513 976 282 989 1 887 992 192 865 460 735 230 127 611 556 45 729 58 12 598 783 732 695 565 718 198 146 308 300 980 617 938 950
510 477 376 78 526 413 189 784 988 356 203 661 114 241 794 551 694 844 253 850 231 766 945 622 69 647 263 557 936 863 567 481 518 303 107 791 803 295 56 909 362 133 994 9
27 106 336 223 311 181 726 749 906 679 305 544 707 632 144 185 55 595 700 650 760 459 479 301 414 515 867 914 29 894 911 918 326 674 386 335 145 939 193

length of the longest wiggle subsequence: 658
time taken: 0.011
--------------------------------
Process exited after 6.624 seconds with return value 0
Press any key to continue . . .
```

Greedy Approach

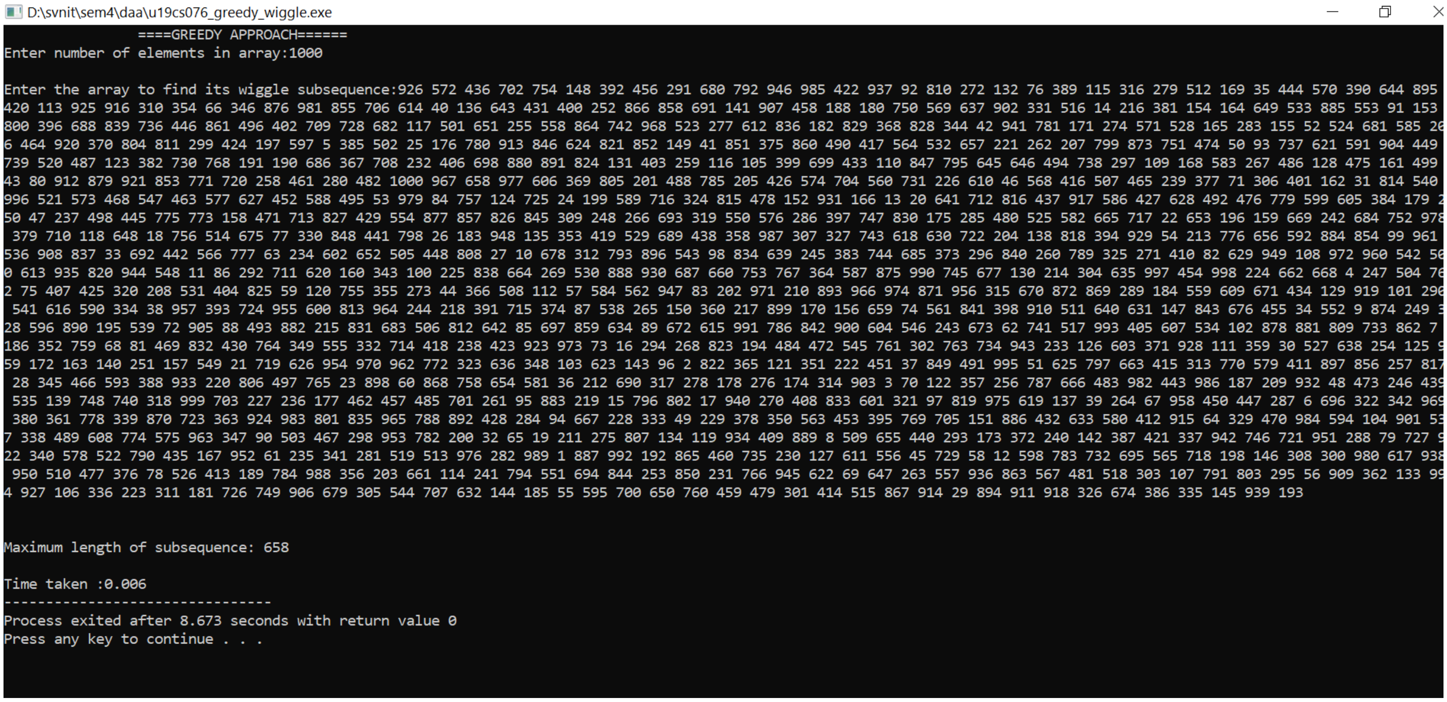


```
#include <iostream>
#include <time.h>
using namespace std;
clock_t begin, end;
double time_;

int wiggle(int arr[],int n)
{
    if (n < 2)
            return n;
    int prevdiff = arr[1] - arr[0];
    int count ;
    if(prevdiff!= 0)
        count=2;
    else
        count=1;
    for (int i = 2; i < n; i++)
    {
        int diff = arr[i] - arr[i - 1];
        if ((diff > 0 && prevdiff <= 0) || (diff < 0 && prevdiff >= 0))
        {
            count++;
            prevdiff = diff;
        }
    }
    return count;
}
int main()
{
    cout<<"\t\t====GREEDY APPROACH======";
    int n;
    cout<<"\nEnter number of elements in array:";
    cin>>n;
    cout<<"\nEnter the array to find its wiggle subsequence:";
    int arr[n];
    for(int i=0;i<n;i++)
        cin>>arr[i];
    begin = clock();
    cout <<  endl<<"\nMaximum length of subsequence: " << wiggle(arr, n) << endl;
    end = clock();
    time_ = ((double)(end-begin)) / CLOCKS_PER_SEC;
    cout<<"\nTime taken :"<<time_;
    return 0;}
```

Output:

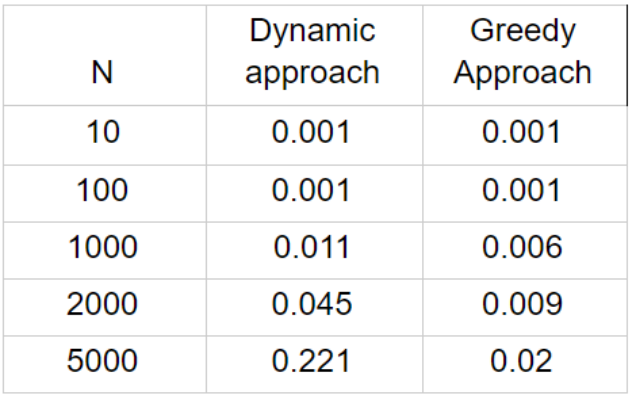


```
                ====GREEDY APPROACH======
Enter number of elements in array:10

Enter the array to find its wiggle subsequence:1 17 5 10 13 15 10 5 16 8


Maximum length of subsequence: 7

Time taken :0.001
--------------------------------
Process exited after 19.02 seconds with return value 0
Press any key to continue . . .
```

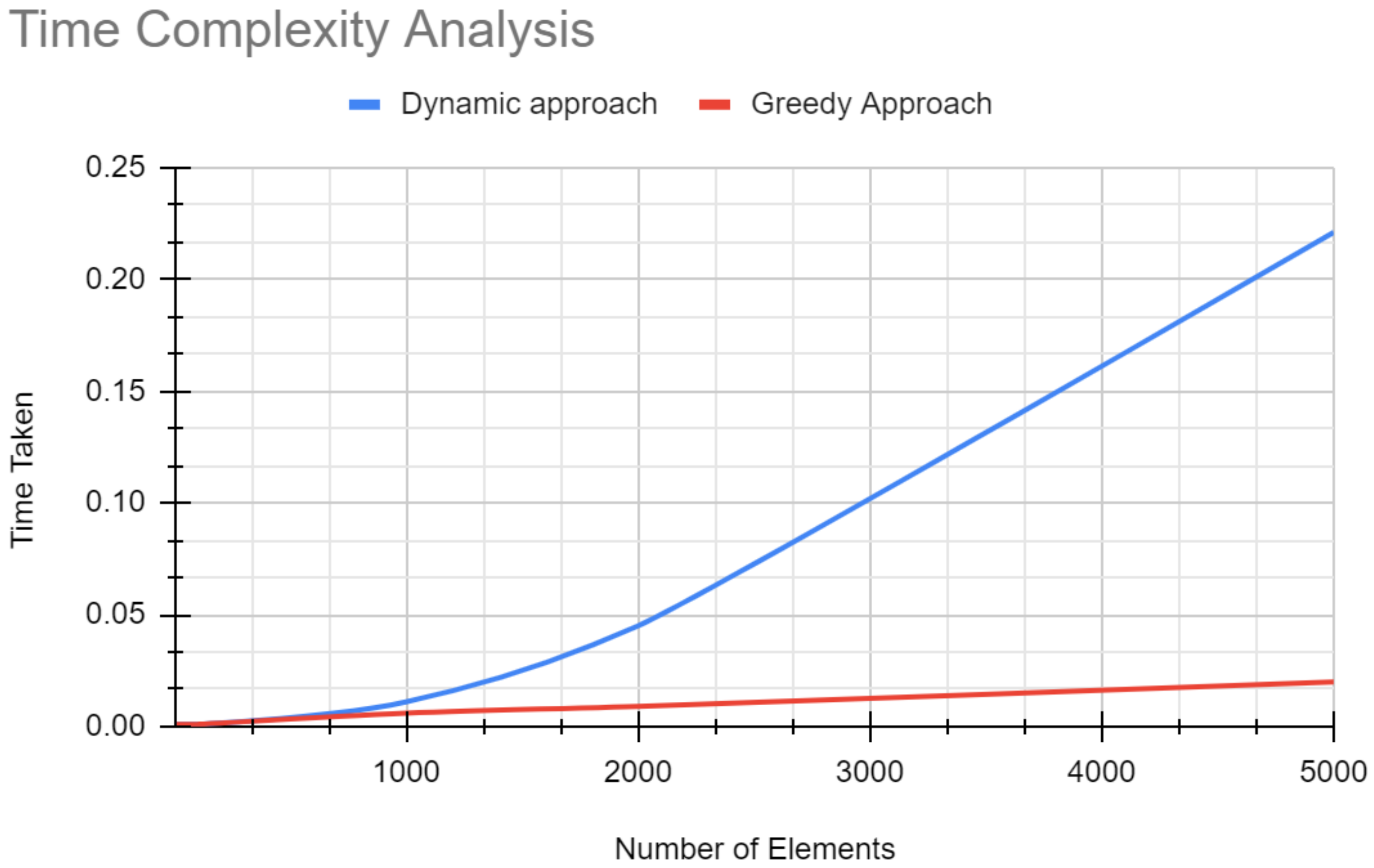


```
                ====GREEDY APPROACH======
Enter number of elements in array:1000

Enter the array to find its wiggle subsequence:926 572 436 702 754 148 392 456 291 680 792 946 985 422 937 92 810 272 132 76 389 115 316 279 512 169 35 444 570 390 644 895
420 113 925 916 310 354 66 346 876 981 855 706 614 40 136 643 431 400 252 866 858 691 141 907 458 188 180 750 569 637 902 331 516 14 216 381 154 164 649 533 885 553 91 153
800 396 688 839 736 446 861 496 402 709 728 682 117 501 651 255 558 864 742 968 523 277 612 836 182 829 368 828 344 42 941 781 171 274 571 528 165 283 155 52 524 681 585 26
6 464 920 370 804 811 299 424 197 597 5 385 502 25 176 780 913 846 624 821 852 149 41 851 375 860 490 417 564 532 657 221 262 207 799 873 751 474 50 93 737 621 591 904 449
739 520 487 123 382 730 768 191 190 686 367 708 232 406 698 880 891 824 131 403 259 116 105 399 699 433 110 847 795 645 646 494 738 297 109 168 583 267 486 128 475 161 499
43 80 912 879 921 853 771 720 258 461 280 482 1000 967 658 977 606 369 805 201 488 785 205 426 574 704 560 731 226 610 46 568 416 507 465 239 377 71 306 401 162 31 814 540
996 521 573 468 547 463 577 627 452 588 495 53 979 84 757 124 725 24 199 589 716 324 815 478 152 931 166 13 20 641 712 816 437 917 586 427 628 492 476 779 599 605 384 179 2
50 47 237 498 445 775 773 158 471 713 827 429 554 877 857 826 845 309 248 266 693 319 550 576 286 397 747 830 175 285 480 525 582 665 717 22 653 196 159 669 242 684 752 978
 379 710 118 648 18 756 514 675 77 330 848 441 798 26 183 948 135 353 419 529 689 438 358 987 307 327 743 618 630 722 204 138 818 394 929 54 213 776 656 592 884 854 99 961
536 908 837 33 692 442 566 777 63 234 602 652 505 448 808 27 10 678 312 793 896 543 98 834 639 245 383 744 685 373 296 840 260 789 325 271 410 82 629 949 108 972 960 542 56
0 613 935 820 944 548 11 86 292 711 620 160 343 100 225 838 664 269 530 888 930 687 660 753 767 364 587 875 990 745 677 130 214 304 635 997 454 998 224 662 668 4 247 504 76
2 75 407 425 320 208 531 404 825 59 120 755 355 273 44 366 508 112 57 584 562 947 83 202 971 210 893 966 974 871 956 315 670 872 869 289 184 559 609 671 434 129 919 101 290
 541 616 590 334 38 957 393 724 955 600 813 964 244 218 391 715 374 87 538 265 150 360 217 899 170 156 659 74 561 841 398 910 511 640 631 147 843 676 455 34 552 9 874 249 3
28 596 890 195 539 72 905 88 493 882 215 831 683 506 812 642 85 697 859 634 89 672 615 991 786 842 900 604 546 243 673 62 741 517 993 405 607 534 102 878 881 809 733 862 7
186 352 759 68 81 469 832 430 764 349 555 332 714 418 238 423 923 973 73 16 294 268 823 194 484 472 545 761 302 763 734 943 233 126 603 371 928 111 359 30 527 638 254 125 9
59 172 163 140 251 157 549 21 719 626 954 970 962 772 323 636 348 103 623 143 96 2 822 365 121 351 222 451 37 849 491 995 51 625 797 663 415 313 770 579 411 897 856 257 817
 28 345 466 593 388 933 220 806 497 765 23 898 60 868 758 654 581 36 212 690 317 278 178 276 174 314 903 3 70 122 357 256 787 666 483 982 443 986 187 209 932 48 473 246 439
 535 139 748 740 318 999 703 227 236 177 462 457 485 701 261 95 883 219 15 796 802 17 940 270 408 833 601 321 97 819 975 619 137 39 264 67 958 450 447 287 6 696 322 342 969
 380 361 778 339 870 723 363 924 983 801 835 965 788 892 428 284 94 667 228 333 49 229 378 350 563 453 395 769 705 151 886 432 633 580 412 915 64 329 470 984 594 104 901 53
7 338 489 608 774 575 963 347 90 503 467 298 953 782 200 32 65 19 211 275 807 134 119 934 409 889 8 509 655 440 293 173 372 240 142 387 421 337 942 746 721 951 288 79 727 9
22 340 578 522 790 435 167 952 61 235 341 281 519 513 976 282 989 1 887 992 192 865 460 735 230 127 611 556 45 729 58 12 598 783 732 695 565 718 198 146 308 300 980 617 938
 950 510 477 376 78 526 413 189 784 988 356 203 661 114 241 794 551 694 844 253 850 231 766 945 622 69 647 263 557 936 863 567 481 518 303 107 791 803 295 56 909 362 133 99
4 927 106 336 223 311 181 726 749 906 679 305 544 707 632 144 185 55 595 700 650 760 459 479 301 414 515 867 914 29 894 911 918 326 674 386 335 145 939 193


Maximum length of subsequence: 658

Time taken :0.006
--------------------------------
Process exited after 8.673 seconds with return value 0
Press any key to continue . . .
```

Time Complexity Analysis



N	Dynamic approach	Greedy Approach
10	0.001	0.001
100	0.001	0.001
1000	0.011	0.006
2000	0.045	0.009
5000	0.221	0.02



Time Complexity for Dynamic Programming Approach : O(N^2)
Space Complexity for Dynamic Programming Approach : O(n)

Time Complexity for Greedy Approach: O(N)
Space Complexity for Greedy Approach : O(1)